Algoritmos Aleatorizados

Lauro Didier Lins

ldl@cin.ufpe.br

UFPE - Brasil

Abril de 2005

Algoritmos Aleatorizados

- *Definições*

  Um *algoritmo aleatorizado* é um *algoritmo* que faz escolhas aleatórias durante sua execução. [Motwani et al, Randomized Algorithms]

  Um *algoritmo aleatorizado* é um *algoritmo* que utiliza *bits aleatórios* como *entrada auxiliar* para guiar o seu comportamento. [www.wikipedia.org]

  Um *algoritmo aleatorizado* é um *algoritmo* que tem acesso a uma fonte de *bits aleatórios* independentes e não-viesados; ele então tem permissão para utilizar estes bits aleatórios para influenciar a sua computação. [Motwani et al, Randomized Algorithms]

Algoritmos Aleatorizados

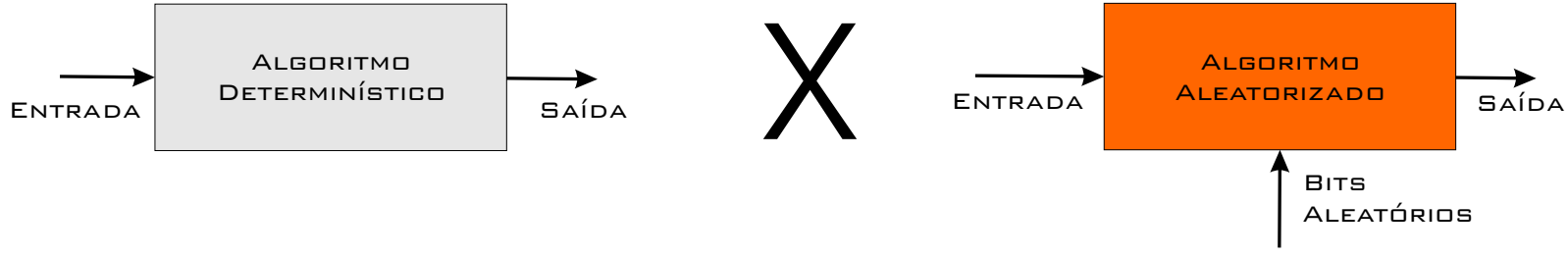
- *Definições*

  Um *algoritmo aleatorizado* é um *algoritmo* que faz escolhas aleatórias durante sua execução. [Motwani et al, Randomized Algorithms]

  Um *algoritmo aleatorizado* é um *algoritmo* que utiliza *bits aleatórios* como *entrada auxiliar* para guiar o seu comportamento. [www.wikipedia.org]

  Um *algoritmo aleatorizado* é um *algoritmo* que tem acesso a uma fonte de *bits aleatórios* independentes e não-viesados; ele então tem permissão para utilizar estes bits aleatórios para influenciar a sua computação. [Motwani et al, Randomized Algorithms]

- *Algoritmo Determinístico* $\times$ *Algoritmo Aleatorizado*



Exemplo 1: Randomized Quicksort

- Considere o problema de ordenar um conjunto S de números. Um algoritmo aleatorizado para resolver tal problema é o seguinte.

  RandQS(S)

  0. Se S é vazio devolva a lista vazia. Se S tem um único elemento devolva a lista com esse único elemento. Senão continue.
  1. Escolha um elemento y de S aleatoriamente, ou seja, todo elemento de S tem a mesma probabilidade de ser escolhido
  2. Comparando todos os elementos de S com y, determine os conjuntos $S_1 = \{x \in S : x < y\}$ e $S_2 = \{x \in S : x > y\}$
  3. Devolva a lista resultante de RandQS(S_1) concatenada com a lista com o elemento y concatenada com a lista resultante de RandQS(S_2).

Exemplo 1: Randomized Quicksort

- Considere o problema de ordenar um conjunto S de números. Um algoritmo aleatorizado para resolver tal problema é o seguinte.

  RandQS(S)

  0. Se S é vazio devolva a lista vazia. Se S tem um único elemento devolva a lista com esse único elemento. Senão continue.
  1. Escolha um elemento y de S aleatoriamente, ou seja, todo elemento de S tem a mesma probabilidade de ser escolhido
  2. Comparando todos os elementos de S com y, determine os conjuntos $S_1 = \{x \in S : x < y\}$ e $S_2 = \{x \in S : x > y\}$
  3. Devolva a lista resultante de RandQS(S_1) concatenada com a lista com o elemento y concatenada com a lista resultante de RandQS(S_2).

- Note no passo 1 que uma escolha aleatória é feita. Isso caracteriza um algoritmo aleatorizado.

Análise do Randomized Quicksort (1)

- Como é usual para algoritmos de ordenação, vamos medir o tempo do RandQS em termos do número de comparações uma vez que esse é o custo dominante em qualquer implementação razoável.

Análise do Randomized Quicksort (1)

- Como é usual para algoritmos de ordenação, vamos medir o tempo do RandQS em termos do número de comparações uma vez que esse é o custo dominante em qualquer implementação razoável.
- Em particular, nosso objetivo é analisar o número esperado de comparações em uma execução do RandQS. Note que todas as comparações são feitas no passo 2 do algoritmo.

Análise do Randomized Quicksort (1)

- Como é usual para algoritmos de ordenação, vamos medir o tempo do RandQS em termos do número de comparações uma vez que esse é o custo dominante em qualquer implementação razoável.
- Em particular, nosso objetivo é analisar o número esperado de comparações em uma execução do RandQS. Note que todas as comparações são feitas no passo 2 do algoritmo.

Def. 1 Denotamos por $S_{(i)}$ o elemento de *rank* i (o i-ésimo menor elemento) no conjunto de números S.

Análise do Randomized Quicksort (1)

- Como é usual para algoritmos de ordenação, vamos medir o tempo do RandQS em termos do número de comparações uma vez que esse é o custo dominante em qualquer implementação razoável.
- Em particular, nosso objetivo é analisar o número esperado de comparações em uma execução do RandQS. Note que todas as comparações são feitas no passo 2 do algoritmo.

Def. 1 Denotamos por $S_{(i)}$ o elemento de *rank* i (o i-ésimo menor elemento) no conjunto de números S.

Def. 2 Definimos a variável aleatória X_{ij} como sendo o número de comparações feitas numa execução do RandQS entre os elementos $S_{(i)}$ e $S_{(j)}$.

Análise do Randomized Quicksort (1)

- Como é usual para algoritmos de ordenação, vamos medir o tempo do RandQS em termos do número de comparações uma vez que esse é o custo dominante em qualquer implementação razoável.
- Em particular, nosso objetivo é analisar o número esperado de comparações em uma execução do RandQS. Note que todas as comparações são feitas no passo 2 do algoritmo.

Def. 1 Denotamos por $S_{(i)}$ o elemento de *rank* i (o i-ésimo menor elemento) no conjunto de números S.

Def. 2 Definimos a variável aleatória X_{ij} como sendo o número de comparações feitas numa execução do RandQS entre os elementos $S_{(i)}$ e $S_{(j)}$.

Prop. 0 O número de comparações esperado é dado por

$$\mathbf{E}\left[\sum_{i=1}^{n}\sum_{j>i}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j>i}^{n} \mathbf{E}[X_{ij}]$$

Prova Isto é uma conseqüência direta da definição de X_{ij} e da linearidade da esperança. □

Análise do Randomized Quicksort (2)

Def. 3 Definimos p_{ij} como sendo a probabilidade de haver comparação entre os elementos $S_{(i)}$ e $S_{(j)}$ numa execução do RandQS.

Análise do Randomized Quicksort (2)

Def. 3 Definimos p_{ij} como sendo a probabilidade de haver comparação entre os elementos $S_{(i)}$ e $S_{(j)}$ numa execução do RandQS.

Prop. 1 A esperança do número de comparações X_{ij} é p_{ij}, formalmente

$$\mathbf{E}[X_{ij}] = p_{ij}.$$

Prova Por construção do RandQS só é possível comparar um par de números no máximo uma vez. Então $X_{ij} \in \{0, 1\}$. Logo a probabilidade p_{ij} de haver comparação entre $S_{(i)}$ e $S_{(j)}$ é exatamente a probabilidade de X_{ij} ser 1, $\mathbf{P}[X_{ij} = 1] = p_{ij}$. ($X_{ij}$ obedece a distribuição de Bernoulli com parâmetro p_{ij}). Logo,

$$\mathbf{E}[X_{ij}] = 1 \times p_{ij} + 0 \times (1 - p_{ij}) = p_{ij}.$$

□

Análise do Randomized Quicksort (2)

Def. 3 Definimos p_{ij} como sendo a probabilidade de haver comparação entre os elementos $S_{(i)}$ e $S_{(j)}$ numa execução do RandQS.

Prop. 1 A esperança do número de comparações X_{ij} é p_{ij}, formalmente

$$\mathbf{E}[X_{ij}] = p_{ij}.$$

Prova Por construção do RandQS só é possível comparar um par de números no máximo uma vez. Então $X_{ij} \in \{0, 1\}$. Logo a probabilidade p_{ij} de haver comparação entre $S_{(i)}$ e $S_{(j)}$ é exatamente a probabilidade de X_{ij} ser 1, $\mathbf{P}[X_{ij} = 1] = p_{ij}$. ($X_{ij}$ obedece a distribuição de Bernoulli com parâmetro p_{ij}). Logo,

$$\mathbf{E}[X_{ij}] = 1 \times p_{ij} + 0 \times (1 - p_{ij}) = p_{ij}.$$

□

Def. 4 Seja T a árvore binária associada a uma execução do RandQS da seguinte forma: a raiz de T é o elemento y escolhido na chamada original a RandQS (primeira chamada); a sub-árvore a esquerda de T contém os elementos de S_1 e a sub-árvore a direita de T contém os elementos de S_2. A estrutura das duas sub-árvores é determinado recursivamente pela execução de RandQS em S_1 e S_2.

Análise do Randomized Quicksort (3)

Prop. 2 O elemento $S_{(i)}$ é comparado com $S_{(j)}$ se e somente se um deles é ancestral do outro em T.

Prova $(\Rightarrow)$ Suponha, que $S_{(i)}$ não é ancestral nem descendente de $S_{(j)}$ em T e que $i < j$. Seja w o ancestral comum a $S_{(i)}$ e a $S_{(j)}$ de maior profundidade na árvore T. Então, $S_{(i)} < w < S_{(j)}$ (caso contrário w não seria o mais profundo ancestral comum). Logo, na chamada a RandQS em que w foi escolhido no passo 1, $S_{(i)}$ e $S_{(j)}$ estavam no conjunto de entrada desta chamada e foram separados: $S_{(i)}$ foi para a sub-árvore da esquerda e $S_{(j)}$ foi para a sub-árvore da direita de forma a que eles nunca serão comparados. $(\Leftarrow)$ Suponha, sem perda de generalidade, que $S_{(i)}$ é ancestral de $S_{(j)}$. Logo, houve uma chamada a RandQS onde $S_{(i)}$ foi escolhido no passo 1 do algoritmo e $S_{(j)}$ fazia parte do conjunto de entrada nesta chamada (caso contrário $S_{(j)}$ não seria descendente de $S_{(i)}$). Então, no passo 2 desta mesma chamada, o elemento $S_{(i)}$ foi comparado ao elemento $S_{(j)}$. $\square$

Análise do Randomized Quicksort (3)

Prop. 2 O elemento $S_{(i)}$ é comparado com $S_{(j)}$ se e somente se um deles é ancestral do outro em T.

Prova $(\Rightarrow)$ Suponha, que $S_{(i)}$ não é ancestral nem descendente de $S_{(j)}$ em T e que $i < j$. Seja w o ancestral comum a $S_{(i)}$ e a $S_{(j)}$ de maior profundidade na árvore T. Então, $S_{(i)} < w < S_{(j)}$ (caso contrário w não seria o mais profundo ancestral comum). Logo, na chamada a RandQS em que w foi escolhido no passo 1, $S_{(i)}$ e $S_{(j)}$ estavam no conjunto de entrada desta chamada e foram separados: $S_{(i)}$ foi para a sub-árvore da esquerda e $S_{(j)}$ foi para a sub-árvore da direita de forma a que eles nunca serão comparados. $(\Leftarrow)$ Suponha, sem perda de generalidade, que $S_{(i)}$ é ancestral de $S_{(j)}$. Logo, houve uma chamada a RandQS onde $S_{(i)}$ foi escolhido no passo 1 do algoritmo e $S_{(j)}$ fazia parte do conjunto de entrada nesta chamada (caso contrário $S_{(j)}$ não seria descendente de $S_{(i)}$). Então, no passo 2 desta mesma chamada, o elemento $S_{(i)}$ foi comparado ao elemento $S_{(j)}$. □

Def. 5 Seja π a única permutação dos nós de T (ou dos elementos de S) cujos elementos satisfazem:

(1) se x é menos profundo do que y em T então x aparece antes de y.

(2) se x e y têm a mesma profundidade em T e $x < y$ então x aparece antes de y.

Análise do Randomized Quicksort (4)

Prop. 3 Há comparação entre os elementos $S_{(i)}$ e $S_{(j)}$ se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre antes na permutação π do que qualquer elemento $S_{(\ell)}$, onde $i < \ell < j$.

Prova Fica como exercício. Dica: Mostrar essa propriedade, pela Proposição 2, é equivalente a mostrar que que $S_{(i)}$ é ancestral ou descendente de $S_{(j)}$ em T se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre antes na permutação π do que qualquer elemento $S_{(\ell)}$, onde $i < \ell < j$. □

Análise do Randomized Quicksort (4)

Prop. 3 Há comparação entre os elementos $S_{(i)}$ e $S_{(j)}$ se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre antes na permutação π do que qualquer elemento $S_{(\ell)}$, onde $i < \ell < j$.

Prova Fica como exercício. Dica: Mostrar essa propriedade, pela Proposição 2, é equivalente a mostrar que que $S_{(i)}$ é ancestral ou descendente de $S_{(j)}$ em T se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre antes na permutação π do que qualquer elemento $S_{(\ell)}$, onde $i < \ell < j$. □

Prop. 4 Todos os elementos $S_{(i)}, \ldots, S_{(j)}$ têm a mesma chance de ser o primeiro a ser escolhido no passo 1 do algoritmo RandQS (ou seja, de ser o primeiro a ocorrer em π).

Prova Enquanto nenhum desses elementos é escolhido, todos eles estão andando juntos nas chamadas recursivas. Então, no momento em que o primeiro elemento dentre os $S_{(i)}, \ldots, S_{(j)}$ foi escolhido no passo 1, todos eram candidatos e tinham a mesma chance de terem sido escolhidos. □

Análise do Randomized Quicksort (4)

Prop. 3 Há comparação entre os elementos $S_{(i)}$ e $S_{(j)}$ se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre antes na permutação π do que qualquer elemento $S_{(\ell)}$, onde $i < \ell < j$.

Prova Fica como exercício. Dica: Mostrar essa propriedade, pela Proposição 2, é equivalente a mostrar que que $S_{(i)}$ é ancestral ou descendente de $S_{(j)}$ em T se e somente se $S_{(i)}$ ou $S_{(j)}$ ocorre antes na permutação π do que qualquer elemento $S_{(\ell)}$, onde $i < \ell < j$. □

Prop. 4 Todos os elementos $S_{(i)}, \ldots, S_{(j)}$ têm a mesma chance de ser o primeiro a ser escolhido no passo 1 do algoritmo RandQS (ou seja, de ser o primeiro a ocorrer em π).

Prova Enquanto nenhum desses elementos é escolhido, todos eles estão andando juntos nas chamadas recursivas. Então, no momento em que o primeiro elemento dentre os $S_{(i)}, \ldots, S_{(j)}$ foi escolhido no passo 1, todos eram candidatos e tinham a mesma chance de terem sido escolhidos. □

Prop. 5 A probabilidade de $S_{(i)}$ ser comparado com $S_{(j)}$ é $p_{ij} = 2/(j - i + 1)$.

Prova Esta probabilidade é, pela Proposição 3, igual a probabilidade de $S_{(i)}$ ou $S_{(j)}$ ocorrer em π antes de $S_{(\ell)}$ para $i < \ell < j$. Mas, pela Proposição 4, todos $S_{(i)}, \ldots, S_{(j)}$ têm a mesma probabilidade, $1/(j - i + 1)$, de ocorrer primeiro em π, logo

$$p_{ij} = 2 \times 1/(j - i + 1) = 2/(j - i + 1).$$

□

Análise do Randomized Quicksort (5)

- Com os resultados obtidos, agora somos capazes de encontrar o número esperado de comparações que o algoritmo aleatorizado RandQS realiza. Combinando as proposições 0, 1 e 5 temos:

$$\mathbf{E}\left[\sum_{i=1}^{n}\sum_{j>i}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j>i}^{n} \mathbf{E}[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i}^{n} p_{ij} = \sum_{i=1}^{n}\sum_{j>i}^{n} 2/(j-i+1) =$$

$$\sum_{i=1}^{n}\sum_{k>1}^{n-i+1} 2/k \leq \sum_{i=1}^{n}\sum_{k=1}^{n} 2/k = 2\sum_{i=1}^{n}\sum_{k=1}^{n} 1/k = 2nH_n,$$

onde H_n é o n-ésimo número harmônico e é definido como $H_n = \sum_{k=1}^{n} 1/k$. Isto é a prova para o seguinte resultado:

Análise do Randomized Quicksort (5)

- Com os resultados obtidos, agora somos capazes de encontrar o número esperado de comparações que o algoritmo aleatorizado RandQS realiza. Combinando as proposições 0, 1 e 5 temos:

$$\mathbf{E}\left[\sum_{i=1}^{n}\sum_{j>i}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j>i}^{n} \mathbf{E}[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i}^{n} p_{ij} = \sum_{i=1}^{n}\sum_{j>i}^{n} 2/(j-i+1) =$$

$$\sum_{i=1}^{n}\sum_{k>1}^{n-i+1} 2/k \leq \sum_{i=1}^{n}\sum_{k=1}^{n} 2/k = 2\sum_{i=1}^{n}\sum_{k=1}^{n} 1/k = 2nH_n,$$

  onde H_n é o n-ésimo número harmônico e é definido como $H_n = \sum_{k=1}^{n} 1/k$. Isto é a prova para o seguinte resultado:

Teo. 1 O número esperado de comparações na execução do algorimo RandQS é no máximo $2nH_n$.

Análise do Randomized Quicksort (5)

- Com os resultados obtidos, agora somos capazes de encontrar o número esperado de comparações que o algoritmo aleatorizado RandQS realiza. Combinando as proposições 0, 1 e 5 temos:

$$\mathbf{E}\left[\sum_{i=1}^{n}\sum_{j>i}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j>i}^{n} \mathbf{E}[X_{ij}] = \sum_{i=1}^{n}\sum_{j>i}^{n} p_{ij} = \sum_{i=1}^{n}\sum_{j>i}^{n} 2/(j-i+1) =$$

$$\sum_{i=1}^{n}\sum_{k>1}^{n-i+1} 2/k \leq \sum_{i=1}^{n}\sum_{k=1}^{n} 2/k = 2\sum_{i=1}^{n}\sum_{k=1}^{n} 1/k = 2nH_n,$$

  onde H_n é o n-ésimo número harmônico e é definido como $H_n = \sum_{k=1}^{n} 1/k$. Isto é a prova para o seguinte resultado:

Teo. 1 O número esperado de comparações na execução do algorimo RandQS é no máximo $2nH_n$.

- Sabe-se que $H_n = \ln n + \Theta(1)$, logo o número de comparações esperado do algoritmo RandQS é $O(n \log n)$.

Randomized Quicksort: Algumas Conclusões

- Note que no desenvolvimento Teorema 1 em nenhum momento fizemos hipóteses sobre sua entrada. O número esperado de comparações depende apenas das escolhas aleatórias feitas pelo algoritmo. Ou seja, o número esperado de comparações é o mesmo qualquer que seja a entrada.

Randomized Quicksort: Algumas Conclusões

- Note que no desenvolvimento Teorema 1 em nenhum momento fizemos hipóteses sobre sua entrada. O número esperado de comparações depende apenas das escolhas aleatórias feitas pelo algoritmo. Ou seja, o número esperado de comparações é o mesmo qualquer que seja a entrada.
- Este comportamento é típico de algoritmos aleatorizados. Seu tempo de execução pode variar mesmo em execuções diferentes de uma mesma entrada.

Randomized Quicksort: Algumas Conclusões

- Note que no desenvolvimento Teorema 1 em nenhum momento fizemos hipóteses sobre sua entrada. O número esperado de comparações depende apenas das escolhas aleatórias feitas pelo algoritmo. Ou seja, o número esperado de comparações é o mesmo qualquer que seja a entrada.

- Este comportamento é típico de algoritmos aleatorizados. Seu tempo de execução pode variar mesmo em execuções diferentes de uma mesma entrada.

- É importante notar que a independência entre *o tempo de execução do algoritmo aleatorizado* e *o valor específico de sua entrada* pode ser uma propriedade valiosa para a *criptografia*. Um determinado algoritmo onde não tem como um “inimigo” fornecer uma entrada em que ele vá se comportar mal pode ser essencial para a segurança de um sistema. Como vimos com o RandQS ele pode até demorar, mas isso independe da entrada, foram suas escolhas aleatórias que “deram azar”.

Randomized Quicksort: Algumas Conclusões

- Note que no desenvolvimento Teorema 1 em nenhum momento fizemos hipóteses sobre sua entrada. O número esperado de comparações depende apenas das escolhas aleatórias feitas pelo algoritmo. Ou seja, o número esperado de comparações é o mesmo qualquer que seja a entrada.

- Este comportamento é típico de algoritmos aleatorizados. Seu tempo de execução pode variar mesmo em execuções diferentes de uma mesma entrada.

- É importante notar que a independência entre *o tempo de execução do algoritmo aleatorizado* e *o valor específico de sua entrada* pode ser uma propriedade valiosa para a *criptografia*. Um determinado algoritmo onde não tem como um “inimigo” fornecer uma entrada em que ele vá se comportar mal pode ser essencial para a segurança de um sistema. Como vimos com o RandQS ele pode até demorar, mas isso independe da entrada, foram suas escolhas aleatórias que “deram azar”.

Def. *(Las Vegas)* Um Algoritmo Aleatorizado que sempre fornece a resposta certa e onde única variação de uma execução para outra é o tempo de execução é chamado do tipo *Las Vegas*. Nestes algoritmos há interesse em estudar a distribuição do tempo de execução do algoritmo.

Randomized Quicksort: Algumas Conclusões

- Note que no desenvolvimento Teorema 1 em nenhum momento fizemos hipóteses sobre sua entrada. O número esperado de comparações depende apenas das escolhas aleatórias feitas pelo algoritmo. Ou seja, o número esperado de comparações é o mesmo qualquer que seja a entrada.

- Este comportamento é típico de algoritmos aleatorizados. Seu tempo de execução pode variar mesmo em execuções diferentes de uma mesma entrada.

- É importante notar que a independência entre *o tempo de execução do algoritmo aleatorizado* e *o valor específico de sua entrada* pode ser uma propriedade valiosa para a *criptografia*. Um determinado algoritmo onde não tem como um "inimigo" fornecer uma entrada em que ele vá se comportar mal pode ser essencial para a segurança de um sistema. Como vimos com o RandQS ele pode até demorar, mas isso independe da entrada, foram suas escolhas aleatórias que "deram azar".

Def. *(Las Vegas)* Um Algoritmo Aleatorizado que sempre fornece a resposta certa e onde única variação de uma execução para outra é o tempo de execução é chamado do tipo *Las Vegas*. Nestes algoritmos há interesse em estudar a distribuição do tempo de execução do algoritmo.

- O Randomized Quicksort é um exemplo de Algoritmo Aleatorizado do tipo *Las Vegas* para o qual estudamos uma das características da distribuição do seu tempo de execução (*i.e. número de comparações*): "a esperança". A variância seria outra característica importante de se estudar, mas que não faremos.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (1)

Def. *(Problema do Corte Mínimo)* Seja G um grafo conexo, não dirigido. Um *corte* em G é um conjunto de arestas cuja remoção resulta em G ser quebrado em dois ou mais componentes. O *corte mínimo em* G é um corte de cardinalidade mínima em G. O *problema de corte mínimo* é encontrar um corte mínimo em G.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (1)

Def. *(Problema do Corte Mínimo)* Seja G um grafo conexo, não dirigido. Um *corte* em G é um conjunto de arestas cuja remoção resulta em G ser quebrado em dois ou mais componentes. O *corte mínimo em* G é um corte de cardinalidade mínima em G. O *problema de corte mínimo* é encontrar um corte mínimo em G.

Def. *(Contração de uma aresta)* Seja $G = (V, E)$ um grafo não dirigido sem laços e $e \in E$ uma aresta entre os vértices u e v. O grafo resultante $G' = (V', E')$ da contração de e em G é definido da seguinte forma:

(1) $V' = (V \backslash \{u, v\}) \cup \{w\}$, onde w é um novo vértice (não presente em G).

(2) As arestas de G com extremidades diferentes de u e v são também arestas em G'.

(3) As arestas de G com apenas uma extremidade em $\{u, v\}$ viram arestas em G' substituindo a extremidade em $\{u, v\}$ por w.

(4) As arestas de G com as duas extremidades em $\{u, v\}$ não aparecem em G'.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (1)

Def. *(Problema do Corte Mínimo)* Seja G um grafo conexo, não dirigido. Um *corte* em G é um conjunto de arestas cuja remoção resulta em G ser quebrado em dois ou mais componentes. O *corte mínimo em* G é um corte de cardinalidade mínima em G. O *problema de corte mínimo* é encontrar um corte mínimo em G.

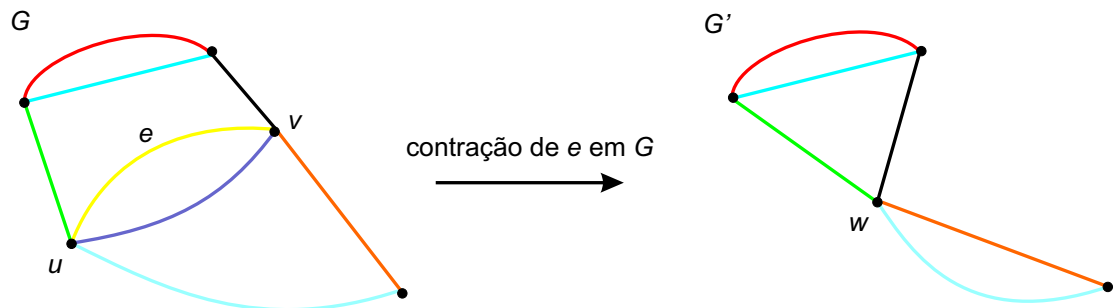
Def. *(Contração de uma aresta)* Seja $G = (V, E)$ um grafo não dirigido sem laços e $e \in E$ uma aresta entre os vértices u e v. O grafo resultante $G' = (V', E')$ da contração de e em G é definido da seguinte forma:

(1) $V' = (V \backslash \{u, v\}) \cup \{w\}$, onde w é um novo vértice (não presente em G).

(2) As arestas de G com extremidades diferentes de u e v são também arestas em G'.

(3) As arestas de G com apenas uma extremidade em $\{u, v\}$ viram arestas em G' substituindo a extremidade em $\{u, v\}$ por w.

(4) As arestas de G com as duas extremidades em $\{u, v\}$ não aparecem em G'.



Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (1)

Def. *(Problema do Corte Mínimo)* Seja G um grafo conexo, não dirigido. Um *corte* em G é um conjunto de arestas cuja remoção resulta em G ser quebrado em dois ou mais componentes. O *corte mínimo em* G é um corte de cardinalidade mínima em G. O *problema de corte mínimo* é encontrar um corte mínimo em G.

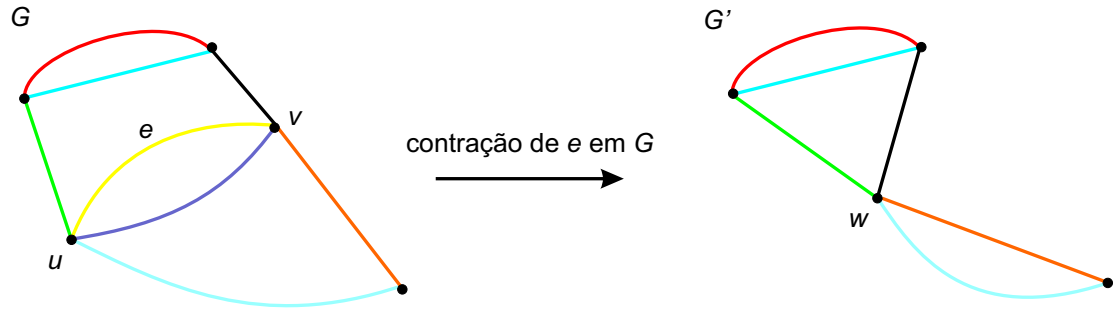
Def. *(Contração de uma aresta)* Seja $G = (V, E)$ um grafo não dirigido sem laços e $e \in E$ uma aresta entre os vértices u e v. O grafo resultante $G' = (V', E')$ da contração de e em G é definido da seguinte forma:

(1) $V' = (V \backslash \{u, v\}) \cup \{w\}$, onde w é um novo vértice (não presente em G).

(2) As arestas de G com extremidades diferentes de u e v são também arestas em G'.

(3) As arestas de G com apenas uma extremidade em $\{u, v\}$ viram arestas em G' substituindo a extremidade em $\{u, v\}$ por w.

(4) As arestas de G com as duas extremidades em $\{u, v\}$ não aparecem em G'.



- Note que o grafo resultante de uma contração de aresta tem um vértice a menos do que o grafo original.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (2)

- Propomos, a seguir, um algoritmo para o problema do corte mínimo que chamamos aqui de RandMC (*Randomized Min-Cut*). Suponha, de agora em diante, que G é um *multigrafo* (*i.e.* grafo que admite múltiplas arestas entre dois vértices) conexo, não dirigido, sem laços, com n vértices e $(n \geq 2)$.

  RandQS(S)

  0. Se G tem apenas 2 vértices u e v, então declare como corte mínimo de G as arestas entre u e v. Senão continue.
  1. Escolha uma aresta e aleatoriamente em G e faça G ser o resultado da contração de e em G depois vá para o passo 0.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (2)

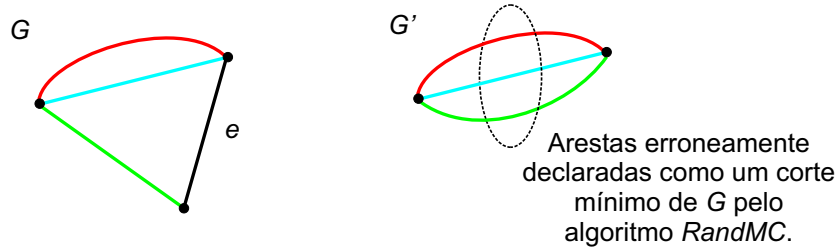
- Propomos, a seguir, um algoritmo para o problema do corte mínimo que chamamos aqui de RandMC (*Randomized Min-Cut*). Suponha, de agora em diante, que G é um *multigrafo* (*i.e.* grafo que admite múltiplas arestas entre dois vértices) conexo, não dirigido, sem laços, com n vértices e $(n \geq 2)$.

  RandQS(S)

  0. Se G tem apenas 2 vértices u e v, então declare como corte mínimo de G as arestas entre u e v. Senão continue.
  1. Escolha uma aresta e aleatoriamente em G e faça G ser o resultado da contração de e em G depois vá para o passo 0.

Prop. 6 O algoritmo RandMC nem sempre devolve um corte mínimo.

Prova O único corte mínimo do grafo a seguir tem cardinalidade 2 e é formado pelas arestas verde e preta, porém o RandMC pode perfeitamente se comportar assim:



□

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (3)

- Por outro lado, apesar de nem sempre retornar um corte mínimo, o seguinte resultado garante que o RandMC sempre retorna um corte de G.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (3)

- Por outro lado, apesar de nem sempre retornar um corte mínimo, o seguinte resultado garante que o RandMC sempre retorna um corte de G.

Prop. 7 Todo corte num grafo contraído em uma aresta é também um corte do grafo original.

Prova exercício □

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (3)

- Por outro lado, apesar de nem sempre retornar um corte mínimo, o seguinte resultado garante que o RandMC sempre retorna um corte de G.

Prop. 7 Todo corte num grafo contraído em uma aresta é também um corte do grafo original.

Prova exercício □

- Vamos, a partir de agora, analisar a chance do algoritmo RandMC encontrar um corte mínimo. . .

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (3)

- Por outro lado, apesar de nem sempre retornar um corte mínimo, o seguinte resultado garante que o RandMC sempre retorna um corte de G.

Prop. 7 Todo corte num grafo contraído em uma aresta é também um corte do grafo original.

Prova exercício □

- Vamos, a partir de agora, analisar a chance do algoritmo RandMC encontrar um corte mínimo. . .

Prop. 8 Se a cardinalidade de um corte mínimo de G é k, então G tem pelo menos $kn/2$ arestas.

Prova Se G tivesse menos de $kn/2$ arestas existiria pelo menos um vértice com grau menor do que k. Logo as arestas incidentes a este vértice seriam um corte de G de cardinalidade menor do que k, ou seja, uma contradição. □

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (3)

- Por outro lado, apesar de nem sempre retornar um corte mínimo, o seguinte resultado garante que o RandMC sempre retorna um corte de G.

Prop. 7 Todo corte num grafo contraído em uma aresta é também um corte do grafo original.

Prova exercício □

- Vamos, a partir de agora, analisar a chance do algoritmo RandMC encontrar um corte mínimo. . .

Prop. 8 Se a cardinalidade de um corte mínimo de G é k, então G tem pelo menos $kn/2$ arestas.

Prova Se G tivesse menos de $kn/2$ arestas existiria pelo menos um vértice com grau menor do que k. Logo as arestas incidentes a este vértice seriam um corte de G de cardinalidade menor do que k, ou seja, uma contradição. □

Def. 6 Seja C um corte mínimo de G. Seja E_i o evento de não escolher uma aresta em C na i-ésima iteração do algoritmo RandMC, para $1 \leq i \leq n-2$.

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (3)

- Por outro lado, apesar de nem sempre retornar um corte mínimo, o seguinte resultado garante que o RandMC sempre retorna um corte de G.

Prop. 7 Todo corte num grafo contraído em uma aresta é também um corte do grafo original.

Prova exercício □

- Vamos, a partir de agora, analisar a chance do algoritmo RandMC encontrar um corte mínimo. . .

Prop. 8 Se a cardinalidade de um corte mínimo de G é k, então G tem pelo menos $kn/2$ arestas.

Prova Se G tivesse menos de $kn/2$ arestas existiria pelo menos um vértice com grau menor do que k. Logo as arestas incidentes a este vértice seriam um corte de G de cardinalidade menor do que k, ou seja, uma contradição. □

Def. 6 Seja C um corte mínimo de G. Seja E_i o evento de não escolher uma aresta em C na i-ésima iteração do algoritmo RandMC, para $1 \leq i \leq n-2$.

Prop. 9 A probabilidade de que uma aresta escolhida aleatoriamente na primeira iteração esteja em C é no máximo $2/n$. Isto é $P[E_1] \geq 1 - 2/n$.

Prova Pela Proposição 8, G tem pelo menos $kn/2$, então a chance de escolher uma aresta de C é no máximo $k/(kn/2)$ (se G tiver mais arestas esta chance diminui). Simplificando, $k/(kn/2) = 2/n$. Mas essa este evento é justamente o complementar do evento E_1 logo $P[E_1] \geq 1 - 2/n$. □

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (4)

Prop. 10 $P[E_i \mid \cap_{j=1}^{i-1} E_j] \geq 1 - 2/(n-i+1)$

Prova Pela Proposição 7 em toda iteração de RandMC temos um grafo cujo corte mínimo tem cardinalidade pelo menos k (caso contrário teríamos um corte no grafo contraído que não seria corte no grafo original uma vez que sua cardinalidade seria $< k$). Como na iteração i o grafo contraído tem $n - i + 1$ vértices, pela Proposição 8 este grafo tem pelo menos $k(n-i+1)/2$ arestas. Daí, dado que $E_1, \ldots, E_{i-1}$ aconteceram, a chance de escolher uma aresta em C na i-ésima iteração é no máximo $k/(k(n-i+1)/2) = 2/(n-i+1)$. Logo, a probabilidade do evento complementar tem o seguinte limitante superior $P[E_i \mid \cap_{j=1}^{i-1} E_j] \leq 1 - 2/(n-i+1)$. □

Exemplo 2: Um Algoritmo Aleatorizado para o Problema do Corte Mínimo (4)

Prop. 10 $P[E_i \mid \cap_{j=1}^{i-1} E_j] \geq 1 - 2/(n-i+1)$

Prova Pela Proposição 7 em toda iteração de RandMC temos um grafo cujo corte mínimo tem cardinalidade pelo menos k (caso contrário teríamos um corte no grafo contraído que não seria corte no grafo original uma vez que sua cardinalidade seria $< k$). Como na iteração i o grafo contraído tem $n-i+1$ vértices, pela Proposição 8 este grafo tem pelo menos $k(n-i+1)/2$ arestas. Daí, dado que $E_1, \ldots, E_{i-1}$ aconteceram, a chance de escolher uma aresta em C na i-ésima iteração é no máximo $k/(k(n-i+1)/2) = 2/(n-i+1)$. Logo, a probabilidade do evento complementar tem o seguinte limitante superior $P[E_i \mid \cap_{j=1}^{i-1} E_j] \leq 1 - 2/(n-i+1)$. □

Teo. 2 A probabilidade do algoritmo RandQS encontrar o corte mínimo C é limitada inferiormente por

$$P[\cap_{i=1}^{n-2} E_i] \geq \frac{2}{n(n-1)}$$

Prova Sabemos da teoria de probabilidade que $P[E_1 \cap E_2] = P[E_1]P[E_2|E_1]$ cuja forma mais geral

$$P[\cap_{i=1}^{n-2} E_i] = P[E_1]P[E_2|E_1]P[E_3|E_1 \cap E_2] \ldots P[E_{n-2}| \cap_{i=1}^{n-3} E_i]$$

aplicamos a seguir junto com a Proposição 10

$$P[\cap_{i=1}^{n-2} E_i] \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2} \left(\frac{n-i-1}{n-i+1}\right) = \frac{\prod_{k=1}^{n-2} k}{\prod_{k=1}^{n-2} k + 2} = \frac{n!}{\frac{(n-2)!}{2!}} = \frac{2}{n(n-1)}$$

□

Um Algoritmo Aleatorizado para o Problema do Corte Mínimo: Algumas Conclusões

- O Teorema 2 nos diz que a chance de descobrir um corte mínimo particular de G (que pode ser o único corte mínimo) é no mínimo $\frac{2}{n(n-1)}$. Isso nos dá também um limitante superior da chance do RandMC errar ao declarar o mínimo que seria $1 - \frac{2}{n(n-1)}$. Logo, a chance do RandMC errar em k execuções independentes para G é dado por $\left(1 - \frac{2}{n(n-1)}\right)^k$. É fácil notar que aumentando k temos uma chance cada vez menor do RandMC não ter encontrado um corte mínimo. Podemos então, rodando RandMC várias vezes e guardando a solução de cardinalidade mínima, chegar a uma solução correta com uma probabilidade tão grande quanto se queira de ter um corte mínimo.

Um Algoritmo Aleatorizado para o Problema do Corte Mínimo: Algumas Conclusões

- O Teorema 2 nos diz que a chance de descobrir um corte mínimo particular de G (que pode ser o único corte mínimo) é no mínimo $\frac{2}{n(n-1)}$. Isso nos dá também um limitante superior da chance do RandMC errar ao declarar o mínimo que seria $1 - \frac{2}{n(n-1)}$. Logo, a chance do RandMC errar em k execuções independentes para G é dado por $\left(1 - \frac{2}{n(n-1)}\right)^k$. É fácil notar que aumentando k temos uma chance cada vez menor do RandMC não ter encontrado um corte mínimo. Podemos então, rodando RandMC várias vezes e guardando a solução de cardinalidade mínima, chegar a uma solução correta com uma probabilidade tão grande quanto se queira de ter um corte mínimo.

- Note a extrema simplicidade do algoritmo RandMC. Ao contrário dos algoritmos determinísticos para o problema do Corte Mínimo que são baseados em *Fluxo em Redes* (um ramo da área Otimização Combinatória) e são consideravelmente mais complexos. Este exemplo particular na verdade ilustra uma característica dos algoritmos aleatorizados em geral: eles são tipicamente os mais simples.

Um Algoritmo Aleatorizado para o Problema do Corte Mínimo: Algumas Conclusões

- O Teorema 2 nos diz que a chance de descobrir um corte mínimo particular de G (que pode ser o único corte mínimo) é no mínimo $\frac{2}{n(n-1)}$. Isso nos dá também um limitante superior da chance do RandMC errar ao declarar o mínimo que seria $1 - \frac{2}{n(n-1)}$. Logo, a chance do RandMC errar em k execuções independentes para G é dado por $\left(1 - \frac{2}{n(n-1)}\right)^k$. É fácil notar que aumentando k temos uma chance cada vez menor do RandMC não ter encontrado um corte mínimo. Podemos então, rodando RandMC várias vezes e guardando a solução de cardinalidade mínima, chegar a uma solução correta com uma probabilidade tão grande quanto se queira de ter um corte mínimo.

- Note a extrema simplicidade do algoritmo RandMC. Ao contrário dos algoritmos determinísticos para o problema do Corte Mínimo que são baseados em *Fluxo em Redes* (um ramo da área Otimização Combinatória) e são consideravelmente mais complexos. Este exemplo particular na verdade ilustra uma característica dos algoritmos aleatorizados em geral: eles são tipicamente os mais simples.

- Existe uma variante do algoritmo RandMC cuja esperança do tempo de execução é consideravelmente menor do que a do melhor algoritmo baseado em Fluxo de Redes (ver seção 10.2 de Motwani et al., Randomized Algorithms).

Algumas Conclusões e Fatos

Def. *(Monte Carlo)* Um Algoritmo Aleatorizado que pode produzir uma resposta errada, mas que somos capazes de limitar a probabilidade deste erro é chamado Algoritmo *Monte Carlo*. Nestes algoritmos o primeiro interesse é em estudar a probabilidade de se obter uma resposta errada (ou, equivalentemente, uma resposta certa).

Algumas Conclusões e Fatos

Def. *(Monte Carlo)* Um Algoritmo Aleatorizado que pode produzir uma resposta errada, mas que somos capazes de limitar a probabilidade deste erro é chamado Algoritmo *Monte Carlo*. Nestes algoritmos o primeiro interesse é em estudar a probabilidade de se obter uma resposta errada (ou, equivalentemente, uma resposta certa).

- O RandMC é um exemplo de algoritmo do tipo *Monte Carlo* para o qual estudamos e encontramos um limitante superior para a probabilidade de erro (acerto). Aqui, não estudamos seu tempo de execução.

Algumas Conclusões e Fatos

Def. *(Monte Carlo)* Um Algoritmo Aleatorizado que pode produzir uma resposta errada, mas que somos capazes de limitar a probabilidade deste erro é chamado Algoritmo *Monte Carlo*. Nestes algoritmos o primeiro interesse é em estudar a probabilidade de se obter uma resposta errada (ou, equivalentemente, uma resposta certa).

- O RandMC é um exemplo de algoritmo do tipo *Monte Carlo* para o qual estudamos e encontramos um limitante superior para a probabilidade de erro (acerto). Aqui, não estudamos seu tempo de execução.
- Um exemplo clássico de algoritmo aleatorizado foi descrito por Miller e Rabin em 1976 e se tratava de um algoritmo para testar a primalidade de um certo número. Na época, nenhum algoritmo determinístico viável para o teste de primalidade era conhecido. Em 2002 um algoritmo determinístico polinomial para o teste de primalidade foi descoberto, porém na prática este teste determinístico não substituiu o teste aleatorizado que se mostra muito mais eficiente e simples.

Algumas Conclusões e Fatos

Def. *(Monte Carlo)* Um Algoritmo Aleatorizado que pode produzir uma resposta errada, mas que somos capazes de limitar a probabilidade deste erro é chamado Algoritmo *Monte Carlo*. Nestes algoritmos o primeiro interesse é em estudar a probabilidade de se obter uma resposta errada (ou, equivalentemente, uma resposta certa).

- O RandMC é um exemplo de algoritmo do tipo *Monte Carlo* para o qual estudamos e encontramos um limitante superior para a probabilidade de erro (acerto). Aqui, não estudamos seu tempo de execução.

- Um exemplo clássico de algoritmo aleatorizado foi descrito por Miller e Rabin em 1976 e se tratava de um algoritmo para testar a primalidade de um certo número. Na época, nenhum algoritmo determinístico viável para o teste de primalidade era conhecido. Em 2002 um algoritmo determinístico polinomial para o teste de primalidade foi descoberto, porém na prática este teste determinístico não substituiu o teste aleatorizado que se mostra muito mais eficiente e simples.

- Se, usando um algoritmo aleatorizado, a probabilidade de erro é 2^{-1000}, a seguinte pergunta filosófica se põe: isto é uma prova matemática? Afinal de contas, a probabilidade de erro é bem menor do que a probabilidade do hardware do computador cometer um erro. O que, em termos práticos, significa considerar esta pequena quantidade uma probabilidade?

Para praticar. . .

Exercício. 1 Considere um algoritmo aleatorizado A do tipo Monte Carlo para um determinado problema. Suponha que seu tempo de execução é no máximo $T(n)$ em qualquer instância de tamanho n e que a chance de A produzir uma resposta certa é pelo menos $\gamma(n)$. Suponha ainda que podemos verificar se uma resposta produzida por A resolve nosso problema em tempo $t(n)$. Mostre como obter um algoritmo aleatorizado B do tipo Las Vegas que sempre produz uma resposta certa e cujo tempo de execução tem esperança no máximo $(T(n) + t(n))/\gamma(n)$.

Para praticar. . .

Exercício. 1 Considere um algoritmo aleatorizado A do tipo Monte Carlo para um determinado problema. Suponha que seu tempo de execução é no máximo $T(n)$ em qualquer instância de tamanho n e que a chance de A produzir uma resposta certa é pelo menos $\gamma(n)$. Suponha ainda que podemos verificar se uma resposta produzida por A resolve nosso problema em tempo $t(n)$. Mostre como obter um algoritmo aleatorizado B do tipo Las Vegas que sempre produz uma resposta certa e cujo tempo de execução tem esperança no máximo $(T(n) + t(n))/\gamma(n)$.

Exercício. 2 Seja $0 < \epsilon_2 < \epsilon_1 < 1$. Considere um algoritmo Monte Carlo que produz uma resposta correta com probabilidade no mínimo $1 - \epsilon_1$, qualquer que seja a entrada. Quantas execuções independentes deste algoritmo são suficientes para elevar este mínimo para $1 - \epsilon_2$, qualquer que seja a entrada?