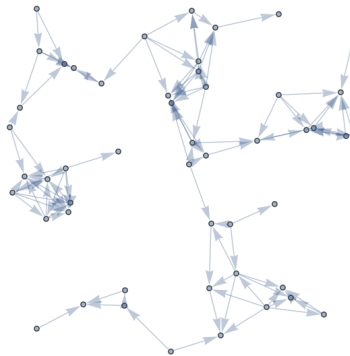


Grafos aleatórios

As funções que discutiremos neste capítulo constroem [grafos aleatórios](#), ou melhor, pseudoaleatórios. (Veja minhas [notas sobre números aleatórios](#).) Esses grafos são muito úteis para testar algoritmos.

Sumário:

- [Um construtor de grafos aleatórios](#)
- [Outro construtor de grafos aleatórios](#)
- [Perguntas e respostas](#)



Um construtor de grafos aleatórios

Nossa primeira função constrói grafos aleatórios com exatamente A arcos. Do ponto de vista do consumo de tempo, essa função não é apropriada para construir [grafos densos](#).

```
/* Para ter acesso à função rand() e à constante RAND_MAX, preciso do ar-
quivo-interface stdlib.h. */

#include <stdlib.h>

/* Esta função constrói um grafo aleatório com vértices 0..V-1 e exatamente
A arcos. A função supõe que  $A \leq V(V-1)$ . Se  $A$  for próximo de  $V(V-1)$ , a
função pode consumir muito tempo. (Código inspirado no Programa 17.7 de
Sedgewick.) */

Graph GRAPHrand( int V, int A ) {
    Graph G = GRAPHinit( V );
    while ( G->A < A ) {
        vertex v = randV( G );
        vertex w = randV( G );
        if ( v != w )
            GRAPHinsertArc( G, v, w );
    }
}
```

```

    }
    return G;
}

/* A função randV() devolve um vértice aleatório do grafo G. Vamos supor que
G->V <= RAND_MAX. */

vertex randV( Graph G) {
    double r;
    r = rand( ) / (RAND_MAX + 1.0);
    return r * G->V;
}

```

A função `randV()` é apenas um invólucro para a função [rand\(.\)](#) da biblioteca `stdlib` que produz um número inteiro (pseudo)aleatórios no intervalo fechado $0..RAND_MAX$. Note que o número r calculado por `randV()` é tal que $0 \leq r < 1$, e portanto a função devolve um inteiro no conjunto $0..V-1$. (Para aplicações pouco exigentes, podemos supor que cada número do intervalo $0..V-1$ tem mais ou menos a mesma probabilidade de ser produzido, especialmente se V for muito menor que `RAND_MAX`.) Veja minha [página sobre números aleatórios](#).

O código de `GRAPHrand()` serve para qualquer representação do grafo — [matriz de adjacências](#) ou [listas de adjacência](#). A função só deve ser invocada com A bem menor que $V*(V-1)$: à medida que A se aproxima desse limite, a execução da função consome cada vez mais tempo, pois `GRAPHinsertArc()` ignora as tentativas de inserir arcos “[repetidos](#)”.

Antes de invocar a função `GRAPHrand()`, o usuário pode usar a função `srand()` da biblioteca `stdlib` para definir uma [semente](#) do gerador de números aleatórios. Enquanto estiver depurando e testando seu programa, o usuário deve preferir um valor fixo da semente, para que a sequência dos números gerados por `rand()` seja sempre a mesma.

Exercícios 1

1. Na função `randV()`, a expressão `rand()/(RAND_MAX+1.0)` pode ser trocada pela expressão `rand()/(RAND_MAX+1)`? Ou pela expressão `rand()/RAND_MAX`? Ou pela expressão `(double)rand()/RAND_MAX`?
2. [Instâncias](#) extremas. Que acontece se `GRAPHrand()` for chamada com A maior que $V*(V-1)$? Que acontece se `GRAPHrand()` for chamada com $G \rightarrow V$ maior que `RAND_MAX+1`?
3. ★ *Permutação aleatória*. Escreva uma função que faça uma permutação aleatória (= *random shuffle*) dos elementos de um vetor `v[0..V-1]` de vértices. (Dica: para k decrescendo de $V-1$ a 1 , escolha um índice aleatório i no intervalo fechado $0..k$ e troque `v[k]` com `v[i]`.)
4. *Atualização da biblioteca*. Acrescente `GRAPHrand()` às bibliotecas `GRAPHmatrix` e `GRAPHlists` que [sugeri em outro capítulo](#).
5. *Testes*. Escreva um programa para testar `GRAPHrand()`. Seu programa deve receber os valores de V e A pela [linha de comando](#), construir um grafo aleatório, e usar a função [GRAPHshow\(.\)](#) para exibir o grafo se V não for muito grande. Cronometre o seu programa quando A estiver próximo de $V*(V-1)$.
6. [Sedgewick 17.69] Escreva um programa que produza, com a mesma probabilidade, cada um dos possíveis grafos com vértices $0..V-1$ e exatamente A arcos.

Outro construtor de grafos aleatórios

A função seguinte `GRAPHrandER()` constrói grafos aleatórios que têm um dado número de arcos *em média*. (O sufixo “ER” é uma referência ao [modelo de Erdős-Rényi](#).) A função é usada principalmente para construir grafos [densos](#).

```
/* Constrói um grafo aleatório com vértices 0..V-1 e número esperado de arcos igual a A. A função supõe que V >= 2 e A <= V*(V-1). (Código inspirado no Program 17.8 de Sedgewick.) */

Graph GRAPHrandER( int V, int A ) {
    double prob = (double) A / (V*(V-1));
    Graph G = GRAPHinit( V );
    for (vertex v = 0; v < V; ++v)
        for (vertex w = 0; w < V; ++w)
            if (v != w)
                if (rand( ) < prob*(RAND_MAX+1.0))
                    GRAPHinsertArc( G, v, w );
    return G;
}
```

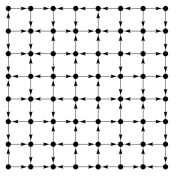
Cada um dos $V*(V-1)$ possíveis arcos é criado com probabilidade `prob`, sendo `prob` calculado de modo que o número esperado de arcos seja `A`.

Exercícios 2

1. A função `GRAPHrandER()` constrói grafos representados por [matriz de adjacências](#) ou por [listas de adjacência](#)?
2. *Instâncias extremas*. Que acontece se a função `GRAPHrandER()` for invocada com `A` maior que $V*(V-1)$? Que acontece se a função for invocada com `V` menor que 2?
3. *Atualização da biblioteca*. Acrescente `GRAPHrandER()` às bibliotecas `GRAPHmatrix` e `GRAPHlists` que [sugeri em outro capítulo](#).
4. *Testes*. Escreva um programa para testar `GRAPHrandER()`. Seu programa deve receber valores de `V` e `A` pela linha de comando, relatar o número de arcos criados, e usar a função `GRAPHshow()` para exibir o grafo.

Exercícios 3

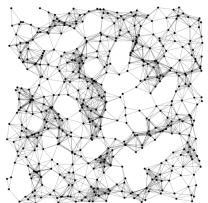

1. *Árvore radicada aleatória*. Escreva uma função `GRAPHrandRootedTree()` que construa uma [árvore radicada](#) aleatória com `V` vértices. (Dica: Comece com uma [permutação aleatória](#) de $0 \ 1 \ \dots \ V-1$. Para `w` de 1 a `V-1`, escolha `v` aleatoriamente em $0..w-1$ e insira um arco `v-w`.) Use a função `GRAPHshow()` para exibir o grafo construído (se `V` não for muito grande). Faça duas versões: uma para [matriz de adjacências](#) e outra para [vetor de listas de adjacência](#).
2. *Floresta radicada aleatória*. Escreva uma função `GRAPHrandRootedForest()` que construa uma [floresta radicada](#) aleatória com `V` vértices e `A` arcos. Use a função `GRAPHshow()` para exibir o grafo construído (se `V` não for muito grande).
3. [Sedgewick fig. 17.13] Escreva uma função que construa um grafo com vértices $0..V-1$ e `V` arcos definidos assim: para cada `v` de 0 a `V-1`, insira um arco `v-w` com `w` escolhido aleatoriamente no intervalo $v+1..v+k$ (sendo `v+1`, `v+2` etc. calculados mó-

- dulo v). Faça testes da sua função. Seu programa de testes deve receber v e k pela linha de comando e usar a função `GRAPHshow()` para exibir o grafo construído.
4. *Torneio aleatório.* Escreva uma função `GRAPHrandTournament()` que constua um [torneio](#) aleatório: para cada par vw de vértices distintos, exatamente um de $v-w$ e $w-v$, com igual probabilidade, é um arco.
 5. *Grafo bipartido dirigido aleatório.* Escreva uma função `GRAPHrandDiBipartite()` que construa um grafo [bipartido dirigido](#) aleatório com V vértices e A arcos de modo que m dos vértices sejam fontes e $V-m$ sejam sorvedouros.
 6. ★ *Grade com orientação aleatória.* Escreva uma função que construa uma [grade dirigida](#) m -por- n e reorienta seus arcos aleatoriamente: cada arco deve ter sua orientação invertida com probabilidade 0.5. Faça testes da sua função. O programa de testes deve receber m e n pela linha de comando, relatar o número de arcos reorientados, e usar a função `GRAPHshow()` para exibir o grafo. 
 7. *Subgrafo gerador aleatório de grade dirigida.* Escreva uma função que construa um [subgrafo gerador](#) aleatório de uma [grade dirigida](#) m -por- n com A arcos em média. Cada um dos arcos da grade deve ser incluído no subgrafo com probabilidade $prob$, sendo $prob$ calculado de modo que o número esperado de arcos seja A . Faça testes da sua função. Seu programa de testes deve receber valores de m , n e A pela linha de comando, relatar o número de arcos incluídos, e usar a função `GRAPHshow()` para exibir o grafo.
 8. *Grade com diagonais aleatórias.* Construa uma [grade dirigida](#) m -por- n . Depois, para cada vértice v , considere o pequeno quadrado cujo canto noroeste é v e acrescente, com probabilidade $prob$, um arco de v até o canto sudeste do quadrado. Faça testes da sua função. Seu programa de testes deve receber valores de m , n e $prob$ pela linha de comando, relatar o número de arcos diagonais acrescentados à grade, e usar a função `GRAPHshow()` para exibir o grafo.
 9. *Arcos antiparalelos aleatórios.* Escreva uma função que receba um grafo G e acrescente, com probabilidade $prob$, um arco [antiparalelo](#) a cada um dos arcos de G . (Alguns livros dizem que um grafo desse tipo é um *mixed graph*.) Faça testes da sua função. Seu programa de testes deve receber $prob$ pela linha de comando, construir um grafo interessante para fazer o papel de G , aplicar a função, relatar o número de arcos acrescentados a G , e usar a função `GRAPHshow()` para exibir o grafo.
 10. *Atualize suas bibliotecas.* Acrescente as funções sugeridas nos exercícios acima à [biblioteca GRAPHmatrix](#) que mencionamos no capítulo *Estruturas de dados para grafos*. Também acrescente as versões apropriadas à [biblioteca GRAPHlists](#). Atualize os correspondentes arquivos-interface.

Exercícios 4: grafos não-dirigidos

A construção de grafos [não-dirigidos](#) aleatórios merece atenção especial. Lembre-se de que uma [aresta](#) é um par de arcos antiparalelos.

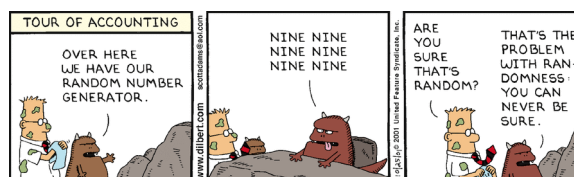
1. Escreva uma função `UGRAPHrand()`, análoga à função `GRAPHrand()`, para construir grafos não-dirigidos aleatórios com v vértices e exatamente E arestas.
2. Escreva uma função `UGRAPHrandER()`, análoga à função `GRAPHrandER()`, para construir grafos não-dirigidos aleatórios com v vértices e número esperado de arestas igual a E .
3. [Sedgewick fig. 17.13] Escreva uma função que construa um grafo não-dirigido com vértices $0..v-1$ e no máximo v arestas inseridas da seguinte maneira: para cada v , escolha w aleatoriamente no intervalo $v-k..v+k$ (com $v-k$, $v-k+1$, etc. calculados módulo v) e insira a aresta $v-w$. Faça testes da sua função. Seu programa de testes deve receber valores de v e k pela linha de comando, relatar o número de arestas inseridas, e usar a função `GRAPHshow()` para exibir o grafo.

4. [Sedgewick 17.73] *Grafos não-dirigido geométricos: pontos próximos.* Escreva uma função `UGRAPHclosePoints()` que escolha v pontos aleatórios no quadrado $[0,1) \times [0,1)$ e em seguida construa um grafo não-dirigido ligando por uma aresta os pontos que estiverem à distância $\leq d$ um do outro. Faça testes da sua função. Seu programa de testes deve receber os valores de v e d pela linha de comando, relatar o número de arestas produzido, e usar a função `GRAPHshow()` para exibir o grafo. 
5. [Sedgewick 17.71] *Subgrafo aleatório de grade não-dirigida.* Escreva uma função que construa um [subgrafo gerador](#) não-dirigido aleatório de uma [grade não-dirigida](#) m -por- n com E arestas. Cada uma das arestas da grade deve ser incluída no subgrafo com probabilidade $prob$, sendo $prob$ calculado de modo que o número esperado de arestas seja E . Faça testes da sua função. Seu programa de testes deve receber valores de m , n e E pela linha de comando, relatar o número de arestas incluídas, e usar a função `GRAPHshow()` para exibir o grafo. Uma variante dessa função pode escolher vértices aleatoriamente e eliminar da grade todas as arestas incidentes a esses vértices. 
6. *Grade com diagonais aleatórias.* Escreva uma variante da [função anterior](#) que construa um [subgrafo gerador](#) não-dirigido da [grade não-dirigida](#) m -por- n e acrescente a ele k arestas aleatoriamente escolhidas dentre as diagonais dos pequenos quadrados da grade. (Basta escolher aleatoriamente k cantos superiores esquerdos dos quadrados.)
7. *Distribuição de graus.* Escreva um programa para estudar a distribuição dos graus em grafos não-dirigido aleatórios. Cada uma das $V(V-1)/2$ possíveis arestas deve ser criada com probabilidade $prob$. Faça testes com $V = 1000$ e $prob = 0.1$. Faça um histograma mostrando o número de vértices que têm grau $0, 1, \dots, V-1$. Faça testes com outros valores de V e $prob$.
8. *Atualize suas bibliotecas.* Acrescente as funções sugeridas nos exercícios acima à [biblioteca GRAPHmatrix](#). Também acrescente as versões apropriadas à [biblioteca GRAPHlists](#). Atualize os correspondentes arquivos-interface.
9. Veja o vídeo [The Mathematics of Randomness: How chance affects our lives way more than you think](#) no YouTube.

Perguntas e respostas

- PERGUNTA: Posso usar a expressão `r = rand() % V` para gerar um número aleatório r no intervalo $0..V-1$?

RESPOSTA: Isso seria razoável se os números produzidos por `rand()` fossem verdadeiramente aleatórios. Ocorre que esses números são apenas *pseudo* aleatórios. Não há qualquer garantia de que os *últimos dígitos* dos números gerados por `rand()` são aleatórios. Em particular, não há qualquer garantia de que o resto da divisão por v seja um número aleatório.



- PERGUNTA: Na documentação das funções, não deveríamos escrever os nomes das variáveis entre aspas? Por exemplo, não deveríamos escrever Esta função constrói um grafo aleatório com ' V ' vértices e ' A ' arcos?

RESPOSTA: Nããão! Afinal, a gente não escreve Hoje encontrei 'Paulo' na faculdade.

- PERGUNTA: Quando escrevemos algo como “o código de `randV()` bla bla” no meio de um texto em português, não deveríamos evitar o par de parênteses depois do nome da função? Afinal, o nome da função é `randV` e não `randV()`.

RESPOSTA: Concordo plenamente. Mas sou obrigado a reconhecer que o par de parênteses é útil para deixar claro que estamos falando de uma função e não de uma variável.

www.ime.usp.br/~pf/algoritmos_para_grafos/

Atualizado em 2017-04-10

Paulo Feofiloff

IME-USP