MAC0328 - Lista 1

Lucas Santos

Agosto 2017

1. Primeiro é importante notar que um grafo com V vértices possui $V * (V - 1)$ arcos diferentes. Para saber quantos grafos diferentes podemos formar com V vértices e A arcos, basta tomar a combinação de $V * (V - 1)$ arcos tomados A a A.

$$C = \frac{n!}{p! * (n - p)!}$$

$$C = \frac{(V * (V - 1))!}{A! * (V * (V - 1) - A)!}$$

2. Cada arco contribui com 1 na soma de graus de saída (e de entrada, também).

3. (a) Para grafos usando matriz de adjacência (complexidade do tempo de execução $\theta(V^2)$):

```
int GRAPHoriented (Graph G) {
    Vertex v, w;
    for (v = 0; v < G->V; v++)
        for (w = v + 1; w < G->V; w++)
            if (G->adj[v][w] != G->adj[w][v])
                return 0;
    return 1;
}
```

(b) Para grafos usando listas de adjacência (complexidade do tempo de execução $O(V + A^2)$):

```
int GRAPHoriented (Graph G) {
    Vertex v;
    link p, q;
    for (v = 0; v < G->V; v++) {
        for (p = G->adj[v]; p != NULL; p = p->next) {
            for (q = G->adj[p->w]; q != NULL && q->w !=
                v; q = q->next);
            if (q == NULL)
                return 1;
        }
    }
    return 0;
}
```

4. Assumindo G um grafo não-dirigido, então o grau de um vértice é o número de arestas no seu leque. Segue abaixo uma implementação em C.

   (a) Para grafos usando matriz de adjacência (complexidade do tempo de execução $\theta(V)$):

```
int GRAPHdeg (Graph G, int v) {
    Vertex w;
    int deg = 0;
    for (w = 0; v < G->V; w++)
        if (G->adj[v][w] == 1)
            deg++;
    return deg;
}
```

   (b) Para grafos usando listas de adjacência (complexidade do tempo de execução $O(V)$):

```
int GRAPHdeg (Graph G, int v) {
    link p;
    int deg = 0;
    for (p = G->adj[v]; p != NULL; deg++, p = p->next);
    return deg;
}
```

5. (a) Para grafos usando matriz de adjacência (complexidade do tempo de execução $\theta(V)$):

```
int GRAPHisolated (Graph G, int v) {
    Vertex w;
    for (w = 0; w < G->V; w++)
        if (G->adj[v][w] == 1 || G->adj[w][v] == 1)
            return 0;
    return 1;
}
```

(b) Para grafos usando listas de adjacência (complexidade do tempo de execução $O(V + A)$):

```
int GRAPHisolated (Graph G, int v) {
    Vertex w;
    link p;
    if (G->adj[v] != NULL)
        return 0;
    for (w = 0; w < G->V; w++)
        for (p = G->adj[w]; p != NULL; p = p->next)
            if (p->w == v)
                return 0;
    return 1;
}
```

6. (a) Para grafos usando matriz de adjacência (complexidade do tempo de execução $\theta(1)$):

```
int GRAPHadj (Graph G, int v, int w) {
    return G->adj[w][v];
}
```

(b) Para grafos usando listas de adjacência (complexidade do tempo de execução $O(V)$):

```
int GRAPHadj (Graph G, int v, int w) {
    Vertex w;
    link p;
    for (p = G->adj[w]; p != NULL; p = p->next)
        if (p->w == v)
            return 1;
    return 0;
}
```

7. O código é o mesmo, independente se a estrutura de dados usadas para o grafo é matriz de adjacência ou listas de adjacência. A complexidade do tempo de execução, entretanto, é diferente. O primeiro é $\theta(V^2)$ enquanto o segundo é $O(V^3)$ e $\Omega(V^2)$. Note que a diferença na complexidade ocorre as diferenças de implementação da função que insere o arco. Segue o código em C:

```
Graph GRAPHknightRand (int n) {
    Graph G = GRAPHinit (n * n);
    int x[8] = { -1, 1, 2, 2, 1, -1, -2, -2 };
    int y[8] = { 2, 2, 1, -1, -2, -2, -1, 1 };
    Vertex l, c;
    Vertex v;
    int i;
    for (v = 0; v < n * n; v++) {
        l = v / n;
        c = v % n;
        for (i = 0; i < 8; i++)
            GRAPHinsertArc (G, v, (l + x[i]) * n + (c +
                y[i]));
    }
    return G;
}
```

8. Segue abaixo uma tabela com os valores de V e A assim como o tempo que foi gasto para construir o grafo G com essas configurações utilizando o construtor aleatório 1.

Table 1: exercício 8

V	**A**	**Tempo gasto (em segundos)**
10.000	10.000	0.48
10.000	10.000	0.49
10.000	1.000.000	0.62
10.000	10.000.000	1.66
10.000	50.000.000	8.56
10.000	95.000.000	34.36
10.000	99.990.000	Não finalizou

9. Abaixo, dois histogramas com valores diferentes de V e A. Os valores eram o que eu esparava, isto é, bem distribuido e com alguns poucos casos destoantes.

Figure 1: $V = 100$, $A = 9000$, média = 49.5, variância = 28.86

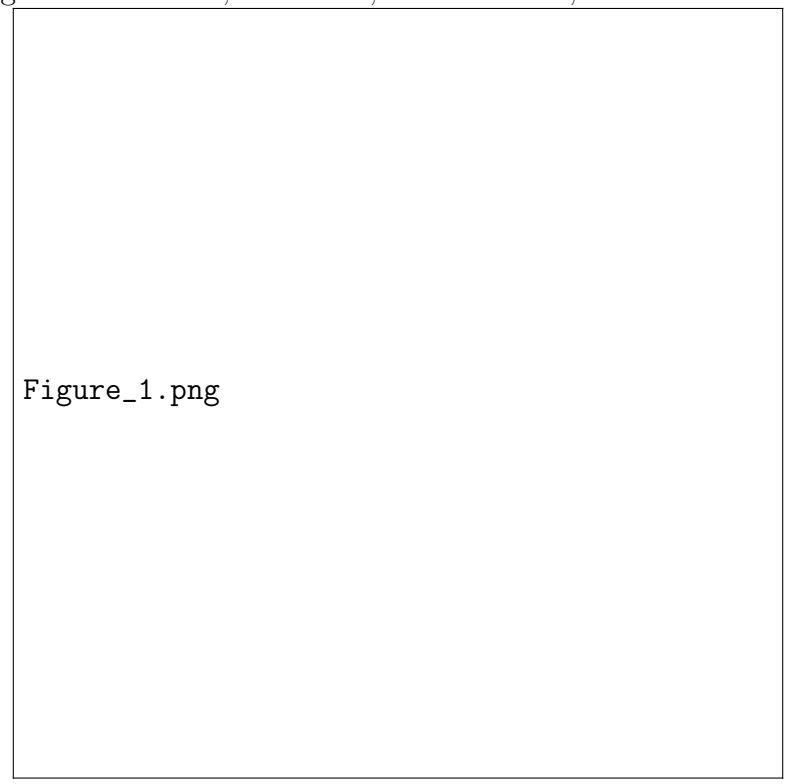
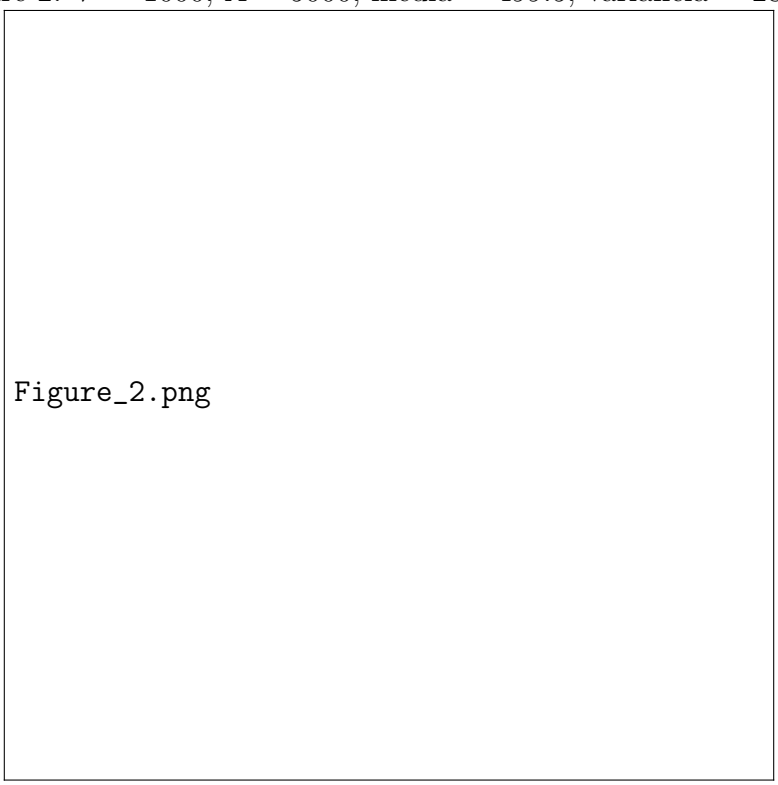


Figure 2: $V = 1000$, $A = 5000$, média = 499.5, variância = 288.67



10. Uma regra que é possível gerar vários grafos é uma que mistura os construtores do rei no xadrez e o de arcos randômicos (tipo 2). A ideia é inserir os arcos do rei no xadrez com uma probabilidade p *(abaixo definida como 0.5)*. A implementação em C é dada por:

```
Graph GRAPHbuildChessKingRand (int n) {
    Graph G = GRAPHinit (n * n);
    Vertex l, c, i, j;
    Vertex v;
    double p = 0.5;
    for (v = 0; v < n * n; v++) {
        l = v / n; // Linha do vrtice atual
        c = v % n; // Coluna do vrtice atual
        for (i = l - 1; i <= l + 1; i++)
            for (j = c - 1; j <= c + 1; j++)
                if (i >= 0 && i < n && j >= 0 && j < n
                    && rand() / (RAND_MAX + 1.0) < p)
```

```
                    GRAPHinsertArc (G, v, i * n + j);
            }
            return G;
        }
```

Esse método constroi um grafo de forma aleatório e isso implica que executando esse método duas vezes não é garantido a construção do mesmo grafo. O grafo π e o grafo da Monalisa não são aleatórios, uma vez que executando o método diversas vezes com a mesma entrada, é garantido que o mesmo grafo será construido.

Realizando alguns experimentos com a regra sugerida (rei no xadrez com probabilidade na existência das arestas), foram feitos 5 experimentos com $p = 0.5$ e $n \in \{5, 10, 20, 50, 100\}$ e foi avaliado algumas propriedades do grafo. São elas:

(a) Grau de entrada e saída: sempre menor ou igual a 6 com uma média próxima de 2. Acredito que tenha sido próximo de 2, e não de 3 (o esperado), devido aos nós nos bordos e quinas.

(b) Embora possível, é bem raro a ocorrência de vértices isolados, fontes ou sorvedouros. Nos casos que foram testados, não houveram nenhuma ocorrência.

(c) Novamente, embora possível, não ocorreram casos onde o grafo é completo ou um torneio.