MCTA003-17 - Análise de Algoritmos
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Prof. Maycon Sambinelli

Grafos

1. Dado um grafo G, o *quadrado* de G é o grafo G^2 tal que $V(G^2) = V(G)$ e, para cada par de vértices $x, y \in V(G)$, a aresta xy vai existir em $E(G^2)$ se e somente se temos $xy \in E(G)$ ou se existe algum outro vértice $w \in V(G)$ tal que as arestas xw e wy existem em $E(G)$. Em outras palavras, em G^2 existe aresta entre todo par de vértices cuja distância é 1 ou 2 em G.

Descreva um algoritmo que recebe um grafo G qualquer e devolve G^2. Tenha certeza que o grafo G^2 não possui arcos paralelos. Analise o tempo de execução se o algoritmo for implementado usando listas de adjacências e se for usando matriz de adjacências. Não é preciso apresentar uma demonstração de correção.

2. O digrafo transposto de um digrafo D é o digrafo D^T tal que $V(D^T) = V(T)$ e $E(D^T) = \{vu : uv \in E(D)\}$. Assim, D^T é D com todas as arestas invertidas. Descreva um algoritmo que recebe um digrafo D qualquer e devolve D^T. Analise o tempo de execução se o algoritmo for implementado usando listas de adjacências e se for usando matriz de adjacências. Não é preciso apresentar uma demonstração de correção.

3. Adapte o algoritmo da Busca em Largura para que o mesmo conte o número de componentes conexas de um grafo e analise a complexidade do algoritmo resultante.

4. Leia a Seção 24.2 das notas de Aula dos Professores Carla N. Lintzmayer e Guilherme O. Mota (veja seção de referências bibliográficas do curso) para aprender sobre o Algoritmo de Busca em Profundidade.

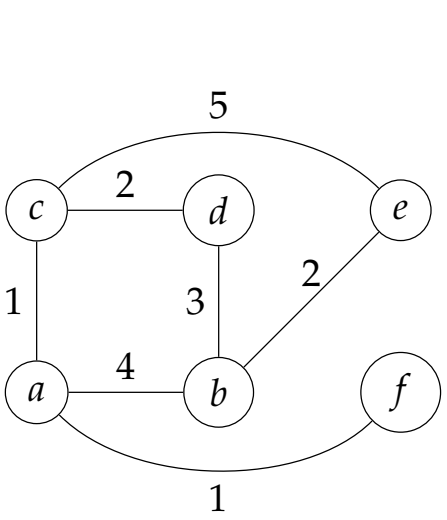
5. Considere o grafo G definido por $V(G) = \{a, b, c, d, e, f, g, h, i\}$ e $E(G) = \{ad, de, ea, ba, bf, fg, gb, cb, ch, hi, ic\}$. Execute a busca em largura sobre G a partir do vértice e. Execute a busca em profundidade sobre G a partir do vértice e. Você pode representar a execução mostrando a evolução do preenchimento de dois vetores, `visitado` e `predecessor`, que são indexados pelos vértices.

6. Prove eu apresente um contraexemplo para a seguinte afirmação: da mesma forma que acontece com a árvore de busca resultante da execução do algoritmo BFS, podemos usar a árvore de busca resultante do algoritmo DFS para encontrar caminhos mínimos da raiz até todos os outros vértices do grafo.

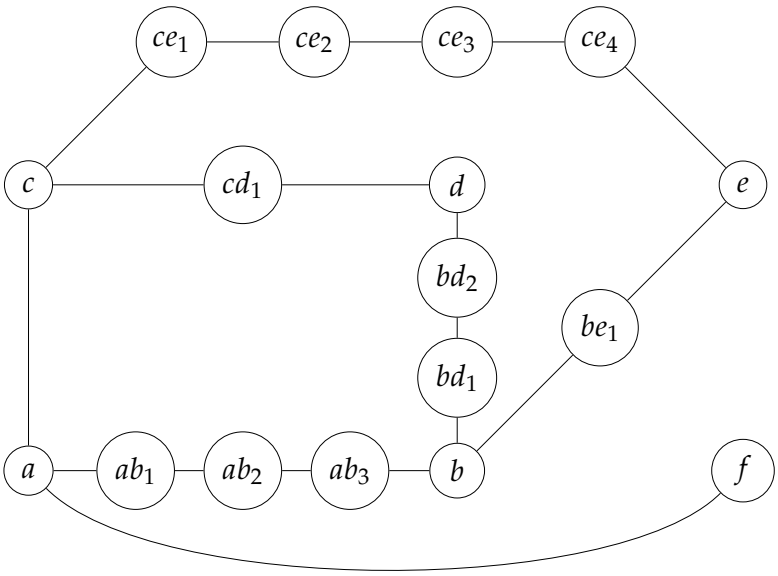
7. Escreva um algoritmo que recebe um grafo G e dois vértices s e v e devolve a sequência de vértices de um sv-caminho em tempo $O(|V(G)| + |E(G)|)$.

8. Dada uma árvore T, o *diâmetro* de T é o valor da maior distância entre quaisquer dois vértices de T. Formalmente, é o número $\max\{dist(u, v) : u, v \in V(T)\}$, onde $dist(u, v)$ é a distância entre u e v em T, em número de arestas. Escreva um algoritmo que, dada T, determine o diâmetro de T em $O(V)$. Justifique a correção do seu algoritmo.

9. Seja $G = (V, E)$ um grafo e $w : E(G) \to \mathbb{Z}_+$ uma função de custo nas arestas tal que $w(e) \geq 1$. Construa o grafo H tal que cada aresta $e \in E(G)$ é substituída por um caminho com $w(e)$ arestas sem custo em H (ou, similarmente, de custo 1). Assim, H possui os mesmos vértices de G e alguns vértices extras. Veja as Figuras 1a e 1b.



(a) Grafo G.



(b) Grafo H.

Figure 1: Exemplos para o exercício 9.

Dado um vértice $s \in V(G)$, qual é o tempo de execução da busca em largura sobre H a partir de s? É possível escrever esse tempo em função de $|V(G)|$ e $|E(G)|$? Para todo $v \in V(G)$, o custo do sv-caminho mínimo em H encontrado pela busca em largura é o mesmo do sv-caminho mínimo em G? Formalize devidamente sua resposta.

1 Novos Exercícios - AA 2023

Dicas 1 Você não precisa formalizar direitinho todos os exercícios da lista (a formalização consome tempo, é burocrático e massante). Formalize alguns até você estar confortável com o processo, afinal é assim que você terá que fazer nas avaliações. Nos outros exercícios, faça apenas anotações rápidas na margem de cada linha.

Dicas 2 Você pode usar qualquer resultado visto em sala de aula sem a necessidade de provar. Apenas referência corretamente o uso de tal resultado.

1. Analise o tempo de execução do algoritmo a seguir.

```
1: Função CHECKSTABILITYNUMBER2(G)
2:     Para cada u ∈ V(G) faça
3:         Para cada v ∈ V(G) \ {u} faça
4:             Se uv ∉ E(G) então
5:                 Para cada w ∈ V(G) \ {u, v} faça
6:                     Se wu ∉ E(G) e wv ∉ E(G) então
7:                         Devolve Falso
8:     Devolve Verdadeiro
```

2. Uma fila de prioridades é uma fila na qual o elementos são processados de acordo com suas prioridades. As operações mais comuns de uma fila de prioridades são:

   INSERIRFILA(Q, e, p) operação que insere um elemento e com prioridade p em uma fila de prioridades Q.

EXTRAIRMINFILA(Q) operação que extrai e retorna o elemento de menor prioridade da fila Q.

EHVAZIA(Q) operação que retorna Verdadeiro se a fila Q é vazia e Falso, caso contrário.

ATUALIZAPRIORIDADEFILA(Q, u, p) operação que atualiza a prioridade do elemento u para p na fila de prioridade Q.

PEGAPRIORIDADEFILA(Q, v) operação que retorna a prioridade do elemento v.

Fila de prioridade é um tipo abstrato de dados e, por conseguinte, pode ser implementado de diversas formas.

(a) Escreva o pseudo-código de uma implementação de filas de prioridades usando vetores e analise o tempo de execução do seu código.

(b) Escreva o pseudo-código de uma implementação de filas de prioridades usando heap binária e analise o tempo de execução do seu código.

(c) Assumindo que a estrutura lista de adjacências foi utilizada para representar o grafo e que vetores foram utilizados para representar a fila de prioridades, analise o tempo de execução do Algoritmo 1.

(d) Assumindo que a estrutura lista de adjacências foi utilizada para representar o grafo e que uma heap binária foi utilizada para representar a fila de prioridades, analise o tempo de execução do Algoritmo 1.

(e) Assumindo que a estrutura matriz de adjacências foi utilizada para representar o grafo e que vetores foram utilizados para representar a fila de prioridades, analise o tempo de execução do Algoritmo 1.

(f) Assumindo que a estrutura matriz de adjacências foi utilizada para representar o grafo e que uma heap binária foi utilizada para representar a fila de prioridades, analise o tempo de execução do Algoritmo 1.

Algorithm 1 Algoritmo de Prim

```
1:  Função PRIM(G, w) ▷ w: E(G) → ℝ é uma função que dá custo às arestas. Qual estrutura de dados você usará para representar isso?
2:      Cria uma fila de prioridades Q
3:      Para cada u ∈ V(G) faça
4:          INSERIRFILA(Q, u, ∞)
5:          pred[u] = Null
6:      Seja s ∈ V(G)
7:      pred[s] = s
8:      ATUALIZAPRIORIDADEFILA(Q, s, 0)
9:      Enquanto não EHVAZIA(Q) faça
10:         u = ExtrairMinFila(Q)
11:         Para cada v ∈ N(u) faça
12:             Se v ∈ Q e w(uv) < PEGAPRIORIDADEFILA(Q, v) então
13:                 pred[v] = u
14:                 ATUALIZAPRIORIDADEFILA(Q, v, w(uv))
```

3. Analise o tempo de execução da função DFS(G, s).

1: **Função** DFS(G, s) ▷ onde G é um grafo e $s \in V(G)$
2: **Para cada** $u \in V(G)$ **faça**
3: $vis[u] = Falso$
4: $pred[u] = Null$
5: $pred[s] = s$
6: DFS_SEARCH(G, s)
7: **Função** DFS_SEARCH(G, u)
8: $vis[u] = Verdadeiro$
9: **Para cada** $v \in N(u)$ **faça**
10: **Se** $vis[v] == Falso$ **então**
11: $pred[v] = u$
12: DFS_SEARCH(G, v)

4. Analise o tempo de execução da função `isNegCycle`.

```
#include <bits/stdc++.h>
using namespace std;

struct Edge { int src, dest, weight; };

struct Graph {
  int V, E;
  struct Edge* edge;
};

struct Graph* createGraph(int V, int E) {
    struct Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge[graph->E];
    return graph;
}

bool isNegCycle(struct Graph* graph, int src) {
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;

    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = graph->edge[j].src;
            int v = graph->edge[j].dest;
            int weight = graph->edge[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
```

```
    }

    for (int i = 0; i < E; i++) {
        int u = graph->edge[i].src;
        int v = graph->edge[i].dest;
        int weight = graph->edge[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            return true;
    }

    return false;
}
```

5. Com relação ao algoritmo BFS visto em sala de aula: prove que se $d[u] < \infty$, então u pertence à árvore T induzida por $pred[]$ e o caminho de s a u em T tem comprimento $d[u]$

6. Prove que se u é um vértice alcançável a partir de um vértice s e uv é uma aresta do grafo, então v também é alcançável.

7. Seja T uma árvore de BFS de um grafo G enraizada em um vértice s. Sejam $L_i = \{u \in V(T) : dist_G(s,u) = i\}$, para $i = 0, 1, \ldots n-1$, os conjuntos com os vértices à distância i da raiz. Prove que se $u \in L_i$, $v \in L_j$ e $|i-j| > 1$, então $uv \notin E(G)$.

8. Mostre que não é possível usar a árvores de DFS para resolver o problema Single Source-Shortes Path, diferentemente do que ocorreu com a árvore de BFS.