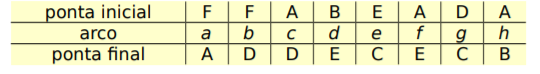
**SIN110 Algoritmos e Grafos - Exercício E4**

**Aluna:** Caroline Lopes Resek

**Matrícula:** 2017010113

1. A tabela abaixo define um grafo com vértices A, B, C, D, E, F. Suponha que, em cada vértice, a lista de adjacências dos arcos que saem do vértice está em ordem alfabética (a, b, c,...). Lista de arcos:



Simule a execução da função de busca em largura. Em que ordem os vértices serão visitados se executarmos uma busca em largura a partir do vértice F? Faça um desenho da arborescência da busca em largura a partir de F até o vértice mais distante



\*Busca em largura do vértice F:

|  |  |  |  |
| --- | --- | --- | --- |
| Vértice | Cor(u) | Predecessor(u) | Dist(u) |
| A | b/c/p | F | 1 |
| B | b/c/p | A | 2 |
| C | b/c/p | D | 2 |
| D | b/c/p | F | 1 |
| E | b/c/p | A | 2 |
| F | b/c/p | - | 0 |

\*Filas:

u = F F- A – D

u = A A - D – E - B

u = D D – E – B – C

u = E E – B – C

u = B B – C

u = C C

\*Ordem de visita: F- A - D - E - B – C

\*Arborescência:



1. Denote por dist(v,w) a distância entre dois vértices v e w num grafo conexo. O diâmetro de um grafo conexo é a valor máximo da expressão dist(v,w) com v e w variando no conjunto de todos os vértices. Escreva uma função que calcule o diâmetro de qualquer grafo conexo.

Função que calcula o diâmetro de qualquer grafo conexo:

Diametro\_grafo\_conexo (G,x)

1. para u ← 1 ate n faca

2. cor[u] ← BRANCO

3. dist[u] ← ∞

4. cor[x] ← CINZA

5. dist[x] ← maior ← 0

6. Q ← Inicializa-Fila (Q,x)

7. enquanto Q ≠ ∅ faça

8. u ← Primeiro-da-Fila (Q)

9. para cada v em Adj[u] faça

10. se cor[v] = BRANCO

11. entao cor[v] ← CINZA

12. dist[v] ← dist[u]+1

13. se dist[v] > maior

14. então maior← dist[v]

15. Insira-na-Fila (Q,v)

16. Remova-da-Fila (Q)

17. cor[u] ← PRETO

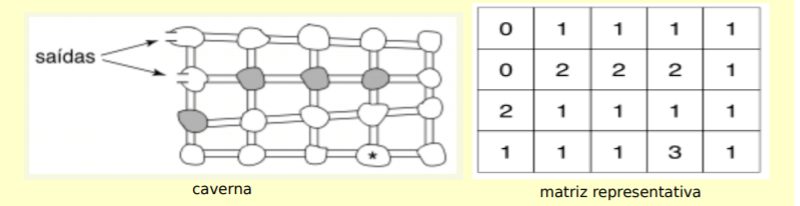
18. devolve maior

Foi criada uma variável maior que receberá a maior distância existente entre o vértice x e os demais vértices do grafo. E essa maior distância, que é o diâmetro do grafo, é retornada pela função ao fim de sua execução.

1. [Adaptado de spoj.com/problems problema 2608 ] “O duende perdido”. Dud, o duende, ficou preso em uma caverna e precisa sair o mais rapidamente possível. A caverna é formada por salões interligados por túneis, na forma de uma grade retangular, com N linhas e M colunas. Alguns dos salões da caverna têm paredes de cristal.

Duendes, como todos sabem, não gostam de ficar em ambientes com qualquer tipo de cristal, pois seus organismos entram em ressonância com a estrutura de cristais, e em casos extremos os duendes podem até mesmo explodir!

Compreensivelmente, Dud não quer entrar em nenhum salão com parede de cristal. A figura abaixo mostra uma caverna com quatro linhas e cinco colunas de salões; os salões cinza têm paredes de cristal. A posição inicial de Dud é indicada com um caractere ‘\*’, conforme ilustra a figura:



Empregando teoria de grafos, escreva um algoritmo em que, dadas a configuração da caverna e a posição inicial de Dud dentro da caverna, calcula qual o número mínimo de salões pelos quais o duende deve passar antes de sair da caverna (não contando o salão em que o duende está inicialmente), mas contando o salão que tem saída para o exterior). Na caverna da figura, o duende deverá percorrer no mínimo 8 salões até encontrar a saída.

A caverna será modelada como uma matriz de duas dimensões, vide figura acima, cujos elementos representam os salões. Um salão que não tem parede de cristal e que tem saída para o exterior da caverna é representado pelo valor 0; um salão que não tem parede de cristal e não tem saída para o exterior é representado pelo valor 1; um salão que tem parede de cristal é representado pelo valor 2; e o salão em que o duende está inicialmente (que não tem saída para o exterior e nem paredes de cristal) é representado pelo valor 3.

Além de projetar, mostre a correção de seu algoritmo e quanto tempo consome para processar a pesquisa.

\*Algoritmo:

Encontra\_MenorCaminho\_Duende(G, x)

1. para u ← 1 ate n faca

2. cor[u] ← BRANCO

3. dist[u] ← ∞

4. cor[x] ← CINZA

5. dist[x] ← 0

6. menor ← ∞

7. Q ← Inicializa-Fila (Q,x)

8. enquanto Q ≠ ∅ faça

9. u ← Primeiro-da-Fila (Q)

10. para cada v em Adj[u] faça

11. se cor[v] = BRANCO e valor[v] ≠ 2

12. entao cor[v] ← CINZA

13. dist[v] ← dist[u]+1

14. se dist[v] < menor e valor[v]=0

15. então menor← dist[v]

16. Insira-na-Fila (Q,v)

17. Remova-da-Fila (Q)

18. cor[u] ← PRETO

19. devolve menor

\*Correção: As condições de parada das linhas 1, 8 e 10 são válidas, logo, a execução do algoritmo tem fim.

O algoritmo funciona, pois, o loop da linha 1 faz uma inicialização das variáveis cor e dist de todos os vértices. O vértice de partida (x) é inicializado com a cor cinza, sua dist com 0 e a variável menor é inicializada com infinito. É inserido na fila Q o vértice x e, na linha 8, entra-se em um loop o qual só terminará quando a fila estiver vazia. Logo após, a variável u é inicializada com o primeiro elemento da fila e na linha 10 entra-se em outro loop que irá percorrer os vértices adjacentes do vértice u. Dentro desse loop é verificado se cor de v é branco e se o valor de v é diferente de dois. Se cor de v for diferente de branco (quer dizer que já foi visitado) e/ou o valor de v for igual a dois (indica que o salão possui parede de cristal), nenhuma operação é executada e o loop continua. Caso contrário, a cor de v fica cinza, o dist de v recebe o dist de u incrementado de um, e, se o dist de v for menor que o número contido na variável menor e o valor de v for igual a zero, significa que foi encontrada uma saída e que a distância foi a menor encontrada até então. Na linha 16, é inserido o vértice v no fim da fila Q e o loop continua. Quando o loop termina é removido o primeiro elemento da fila, a cor de u fica preto e o loop da linha 8 continua enquanto houver elementos na fila. Por fim, é retornado o número mínimo de salões que o duende deverá percorrer para sair da caverna são e salvo.

\*Tempo de execução:

|  |  |
| --- | --- |
| Encontra\_MenorCaminho\_Duende(G, x) | Contagem |
| 1. para u ← 1 ate n faca | n+1 |
| 2. cor[u] ← BRANCO | n |
| 3. dist[u] ← ∞ | n |
| 4. cor[x] ← CINZA | 1 |
| 5. dist[x] ← 0 | 1 |
| 6. menor ← ∞ | 1 |
| 7. Q ← Inicializa-Fila (Q,x) | 1 |
| 8. enquanto Q ≠ ∅ faça | n+1 |
| 9. u ← Primeiro-da-Fila (Q) | n |
| 10. para cada v em Adj[u] faça | v+1 |
| 11. se cor[v] = BRANCO e valor[v] ≠ 2 | v |
| 12. entao cor[v] ← CINZA | v |
| 13. dist[v] ← dist[u]+1 | v |
| 14. se dist[v] < menor e valor[v]=0 | v |
| 15. então menor← dist[v] | v |
| 16. Insira-na-Fila (Q,v) | v |
| 17. Remova-da-Fila (Q) | n |
| 18. cor[u] ← PRETO | n |
| 19. devolve menor | 1 |

Para o cálculo foi considerado número de vértices (n) e número de adjacentes (v).

F(t)= n+1+n+n+1+1+1+1+n+1+n+v+1+v+v+v+v+v+v+n+n+1

F(t)= 7n + 7v + 8

\*Complexidade assintótica: O (n+v).

Logo, o consumo de tempo do algoritmo é de O (n+v).