Estrutura de Dados

Autoria: Carlos Eduardo Cayres | 4º semestre

Tema 08

Grafos e suas Aplicações

Anhanguera

Tema 08

Grafos e suas Aplicações

Autoria: Carlos Eduardo Cayres

Como citar esse documento:

CAYRES, Carlos Eduardo. *Estrutura de Dados*: Grafos e suas Aplicações. Caderno de Atividades. Anhanguera Educacional: Valinhos, 2014.

Índice

CONVITEÀLEITURA

Este tema apresenta a estrutura de dados do tipo grafos, uma das estruturas de programação mais utilizadas em programação.

Grafo pode ser basicamente definido como um conjunto de vértices (ou nós) e um conjunto de arestas (ou arcos), sendo que cada aresta em um grafo é representada por um par de vértices.

Diferentemente de outros ramos da matemática, que muitas vezes apresentam trabalhos com base somente teórica, a teoria dos grafos é focada em problemas práticos do cotidiano. Seu embasamento está direcionado ao estudo de objetos combinatórios (grafos), que representam modelos para problemas em diversos ramos da matemática, computação, engenharia, indústria, entre outros.

Neste tema, apresentaremos conceitos de grafos e introduziremos seu uso em C com vários exemplos práticos. Para finalizar, teremos alguns exercícios para reforçar o conteúdo estudado referente à estrutura de dados do tipo grafos.

PORDENTRODOTEMA

Grafos e suas Aplicações

Uma definição simples de grafo seria a seguinte: um conjunto de vértices (ou nós) e um conjunto de **arestas** (ou arcos), sendo que cada aresta em um grafo é representada por um par de vértices. Um grafo G=(V,A) consiste em:

1. Conjunto finito de **vértices** V. Tais elementos de V são chamados vértices do grafo G.
2. Conjunto finito A de pares não ordenados de V. Tais elementos de A são chamados arestas do grafo G.

A Figura 8.1 a seguir apresenta vários exemplos de grafos, identificados pelas letras a, b, c e d:

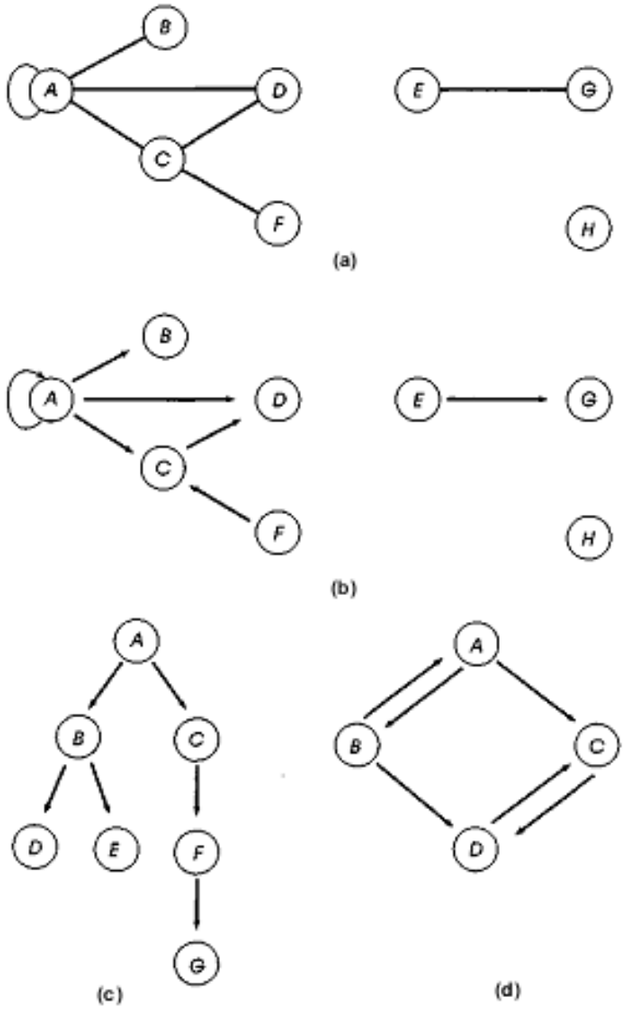


Figura 8.1: Exemplos de grafos: (a), (b), (c) e (d)

Fonte: TENENBAUM; LANGSAM; AUGENSTEIN (1995).

A sequência de nós é {A,B,C,D,E,F,G,H}, e o conjunto de arcos é {(A,B), (A,D), (A,C), (C,D), (C,F), (E,G), (A,A)}. Se os pares de nós que formam os arcos forem pares ordenados, diz-se que o grafo é um grafo orientado (ou dígrafo). As Figuras 1b, c e d ilustram três dígrafos. As setas entre os nós representam arcos. A ponta de cada seta representa o segundo nó no par ordenado de nós que forma um arco, e o final de cada seta representa o

> primeiro nó no par. O conjunto de arcos do grafo da Figura 1b é {<A,B>, <A,C>, <A,D>, <C,D>, <F,C>, <E,G>, <A,A>}. Observe que um grafo não precisa ser uma árvore (Figuras 1a, b e d), mas uma árvore tem de ser um grafo (Figura 1c). Observe também que um nó não precisa ter arcos associados a ele (nó H nas Figuras 1a e b). (TENENBAUM; LANGSAM; AUGENSTEIN, 1995)

Dizemos que um nó x qualquer incide no arco y somente se x representar no par ordenado de nós que forma y um dos dois nós. Neste caso, podemos dizer que y incide em x. O número de arcos que incidem em um nó determina o seu grau. Pode-se observar também que:

3. O número de arcos que um nó tem como cabeça representa seu grau de entrada.

4. O número de arcos que um nó tem como terminação de seta representa seu grau de saída.

Como exemplo, temos o nó A na Figura 8.1 (d), que tem grau 3, sendo grau de entrada 1 e grau de saída 2.

> Um caminho de comprimento k do nó a ao nó b é definido como uma sequência de k + 1 nós n1 , n2 ,..., nk , tal que n = a, nk = b e adjacent(ni + i) é true para todo i entre 1 e k. Se, para algum inteiro k, existir um caminho de comprimento k entre a e b, existirá um caminho de a até b. Um caminho de um nó para si mesmo é chamado ciclo. Se um grafo contiver um ciclo, ele será cíclico; caso contrário, será acíclico. Um grafo acíclico orientado é chamado dag, uma aglutinação das iniciais de *directed acyclic graph*. (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

Aplicações com Grafos

Busca em profundidade

Nos algoritmos de busca em profundidade, as arestas são percorridas partindo do vértice v que tenha sido descoberto por último e que tenha arestas não descobertas a partir dele.

Depois que todas as arestas partindo de v forem descobertas, uma nova busca se inicia para percorrer as arestas partindo do vértice a partir do qual v foi descoberto. Um busca em profundidade poderá gerar algumas árvores, dependendo do número de origens a partir das quais a busca é repetida.

Na busca em profundidade, assim como na busca em largura, os vértices são coloridos indicando seu estado.

POR DENTRO DO TEMA

Exemplo 1 - Busca em profundidade

```
#include <iostream>
#include <map>
#include <list>
#include <string>
using namespace std;
const string P = “preto”;
const string B = “branco”;
const string C = “cinza”;

typedef list<int> listaAdjacencias;
typedef map<int, listaAdjacencias> grafo;
map<int, string> cor;
list<int> ordemTopologica;
void visitarNodo(grafo G, int u)
{
        cor[u] = C;

        for(listaAdjacencias::iterator i = G[u].begin(); i != G[u].end(); i++)
        {
                if ( cor[ *i ] == B )
                        visitarNodo(G, *i);
        }
        cor[u] = P;
        ordemTopologica.push_back(u);
}
```

POR DENTRO DO TEMA

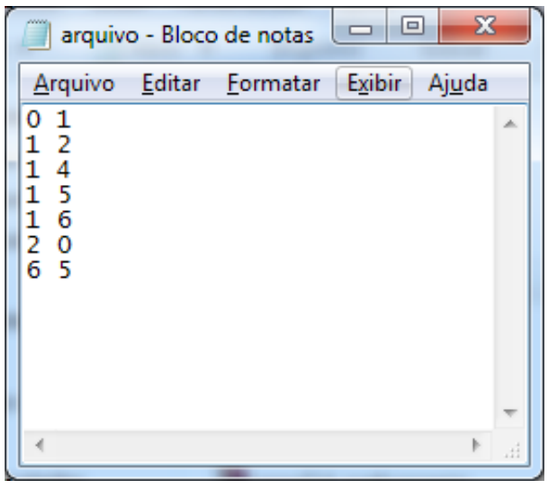
```
void buscaProfundidade(grafo G)
{
        for(grafo::iterator i = G.begin(); i != G.end(); ++i)
                cor[i->first] = B;

        for(grafo::iterator i = G.begin(); i != G.end(); ++i)
        {
                if ( cor[i->first] == B )
                {
                        visitarNodo(G, i->first);
                }
        }
}
int temCiclo(grafo G)
{
        list<int>::iterator i, j;

        buscaProfundidade(G);

        for(i = ordemTopologica.begin(); i != ordemTopologica.end(); i++)
        {
                j = i;
                for(++j; j != ordemTopologica.end(); j++)
                        if (  find(G[*i].begin(), G[*i].end(), *j) != G[*i].end()  )
                                return(0);
        }
```

```
        return(1);
}

int main(int argc, char *argv[])
{
        FILE *arquivo;

        grafo G;
        int no, aresta;

        if( (arquivo = fopen (“c:\\grafos\\arquivo.txt”, “r”)) == NULL )
        {
                printf(“\n\n Erro na leitura do arquivo.\n”);
                exit(1);
        }

        while( !feof(arquivo) )
        {
                fscanf(arquivo,”%d %d”, &no, &aresta);

                if ( G[no].empty() )
                        G[no] = listaAdjacencias();
                if ( G[aresta].empty() )
                        G[aresta] = listaAdjacencias();
                G[no].push_back(aresta);
        }
```

```
        fclose(arquivo);

        for(grafo::iterator i = G.begin(); i != G.end(); ++i)
                if ( ! (i->second).empty() )
                        (i->second).sort();

        if ( temCiclo(G) == 1 )
                printf("Grafo aciclico.\n");
        else
                printf("Ha ciclos no grafo.\n");

        system("PAUSE");
    return(0);
}
```

Arquivo para teste – arquivo.txt

Antes de executar o programa anteriormente descrito, deve-se criar o arquivo a seguir na pasta **c:\grafos\arquivo.txt**.



Lista de adjacências

Uma lista de adjacências em grafos pode ser definida como a representação de todas as arestas ou arcos de um grafo em uma lista.

Em grafos não direcionados, as entradas são conjuntos de dois nós contendo as extremidades da aresta correspondente. No caso de grafos dirigidos, a entrada é uma tupla de dois nós, representando origem e destino de um arco qualquer. Na maioria das representações, as listas de adjacências não são ordenadas.

Exemplo 2 - Lista de adjacências

```
#include <iostream>
#include <stdio.h>
#define TOT_VERT 8

typedef struct itens{
        int cmp;
        struct itens* prox;
}itens;

itens lista[TOT_VERT+1];

void Listar(itens *lista);
void Insere(itens *lista, int a, int b);

int main(int argc, char *argv[]){
  int i,a,b;

  FILE *fp;
```

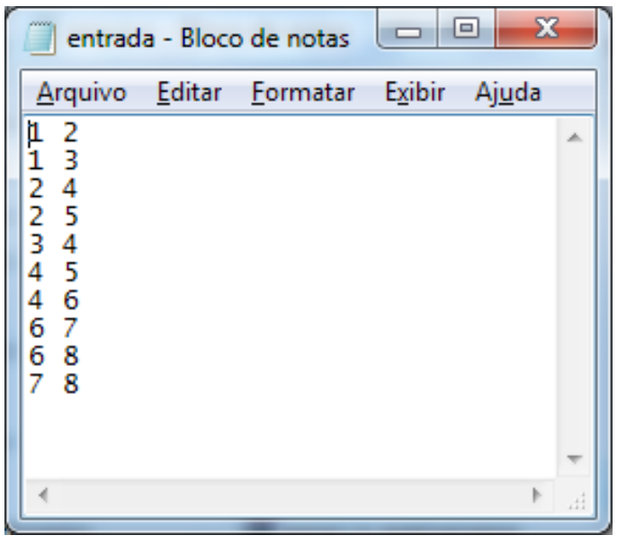
```
  fp = fopen("c:\\grafos\\entrada.txt","r");
  if (!fp) {
    printf("Erro na tentativa de abrir o arquivo %s.\n");
    return 0;
  }

  for(i=1; i<=TOT_VERT; i++){
    lista[i].cmp = 0;
    lista[i].prox = NULL;
  }

  fscanf(fp,"%d %d", &a, &b);
  while (!feof(fp)) {
    Insere(lista,a,b);
    Insere(lista,b,a);
    fscanf(fp,"%d%d", &a, &b);
  }

  Listar(lista);
  system("PAUSE");
}

void Listar(itens *lista){
  int i;
  itens * Temporaria;
  for(i=1; i<=TOT_VERT; i++) {
```

```
    Temporaria = lista[i].prox;
    printf("%2d: (%d) ==>", i, lista[i].cmp);
    while (Temporaria != NULL) {
      printf("%d  ", Temporaria->cmp);
      Temporaria = Temporaria->prox;
    }
    printf("\n");
  }
}
void Insere(itens *lista, int a, int b){
  itens *aux;
  itens *Temporaria;

  aux = (itens*) malloc((int)sizeof(itens));
  aux->cmp = b;
  aux->prox = NULL;

  lista[a].cmp++;
  if(lista[a].prox == NULL)
    lista[a].prox = aux;
  else{
    Temporaria = lista[a].prox;
    if (Temporaria->cmp > b) {
      aux->prox = Temporaria;
      lista[a].prox = aux;
    }
```

```
    else if (Temporaria->prox == NULL) {
      aux->prox = Temporaria->prox;
      Temporaria->prox = aux;
    }
    else {
      while((Temporaria->prox != NULL) &&(Temporaria->prox->cmp < b))
         Temporaria = Temporaria->prox;
      aux->prox = Temporaria->prox;
      Temporaria->prox = aux;
    }
  }
}
```

Arquivo para teste – arquivo.txt

Antes de executar o programa anteriormente descrito, deve-se criar o arquivo a seguir na pasta **c:\grafos\ entrada.txt**.



Menor caminho

> Num grafo ponderado, ou rede, deseja-se frequentemente achar o menor caminho entre dois nós, s e t. O menor caminho é definido como um caminho de s até t, de modo que a soma dos pesos dos arcos do caminho seja minimizada. Para representar a rede, presumimos uma função de peso, de modo que weight(i,j) seja o peso do arco de i a j. Se não existir um arco de i até j, weight(i,j) será definida com valor arbitrariamente grande para indicar o custo infinito (ou seja, a impossibilidade) de seguir diretamente de i até j. (TENENBAUM; LANGSAM; AUGENSTEIN, 1995).

Um dos algoritmos mais utilizados para calcular o caminho de custo mínimo entre vértices de um grafo é o Algoritmo de Dijkstra. Sua execução parte da escolha de um vértice para ser a raiz da busca. Partindo do vértice escolhido, o algoritmo calcula o custo mínimo entre este vértice e os outros vértices do grafo. Apesar de ser um algoritmo simples, apresenta um bom desempenho na execução. Entretanto, caso existam arcos com valores negativos, o algoritmo não garante exatidão da solução.

Na busca do custo mínimo, o algoritmo parte de uma estimativa inicial e vai ajustando esta estimativa.

Exemplo 3 – Menor caminho

```
#include<stdio.h>
#include<stdlib.h>
#define DIM 7
//Variáveis globais
int Grafo[DIM][DIM],Grafo1[DIM][DIM],M_c[DIM][DIM],M_p[DIM][DIM],Conectividade[DIM][DIM],op,ori,des;
void CarreGrafoar(int m[DIM][DIM], int v)
{
  int i,j;
  i=j=0;
    while(i!=7)
      {
          while(j!=7)
            {
```

```
                m[i][j]=v;
                j++;
            }
        j=0;
        i++;
    }
}
void Mostrar(int m[DIM][DIM])
{
  int i,j,k;
  system("CLS");
  i=j=0;
  k=1;
  printf("      ");
    while(k!=8)
      {
        printf("  %d  ",k);
        k++;
      }
  printf("  \n\n\n  ");
  k=1;
    while(i!=7)
      {
        printf("  %d  ",k);
          while(j!=7)
            {
```

```
                printf("  %d  ",m[i][j]);
                j++;
              }
        i++;
        j=0;
        k++;
        printf("  \n\n  ");
      }
}
void Mostrar_1(int m[DIM][DIM])
{
  int i,j,k;
  i=j=0;
  k=1;
  system("CLS");
  printf("    ");
    while(k!=8)
      {
        printf("%d   ",k);
        k++;
      }
  k=1;
  printf("\n\n\n");
    while(i!=7)
      {
        printf("%d   ",k);
```

```
            while(j!=7)
              {
                printf("%d   ",(m[i][j]+1));
                j++;
              }
          i++;
          j=0;
          k++;
          printf("\n\n");
        }
}
void M_conectar(int m[DIM][DIM], int a, int b, int v)
{
  m[a-1][b-1]=v;
}

int M_adjacente(int m[DIM][DIM], int a, int b)
{
  if(m[a-1][b-1] == 0)
    {
      return(0);
    }
  else
    {
      return(1);
    }
```

```
}
void M_custo(int Grafo[DIM][DIM], int c[DIM][DIM], int p[DIM][DIM])
{
   int i,j,k;
   i=j=k=0;
      while(i!=7)
         {
               while(j!=7)
                  {
                        if(Grafo[i][j]!=0)
                           {
                              c[i][j]=Grafo[i][j];
                           }
                        else
                           {
                              c[i][j]=99;
                           }
                     j++;
                  }
            i++;
            j=0;
         }
   i=0;
      while(i!=7)
         {
            c[i][i]=0;
```

```
            i++;
        }
    i=0;
    j=k=1;
      while(k!=7)
        {
            while(i!=7)
              {
                  while(j!=7)
                    {
                        if((c[i][k] + c[k][j]) < c[i][j])
                          {
                            c[i][j] = c[i][k] + c[k][j];
                            p[i][j] = k;
                          }
                      j++;
                    }
                i++;
                j=0;
              }
          k++;
          i=0;
        }
}
void Mostrar_caminho(int m[DIM][DIM], int a, int b)
{
```

POR DENTRO DO TEMA

```
   int k,d,r;
   k=((m[a][b])+1);
      if(k!=0)
        {
          Mostrar_caminho(m, a, k-1);
          printf("%d => ",k);
          Mostrar_caminho(m, k-1, b);
        }
}
void M_conectividade(int m[DIM][DIM], int cd[DIM][DIM])
{
   int i,j,k;
      for(i=0;i<=6;i++)
        {
             for(j=0;j<=6;j++)
               {
                  cd[i][j]=m[i][j];
               }
        }
      for(k=0;k<=6;++k)
        {
             for(i=0;i<=6;++i)
               {
                    for(j=0;j<=6;++j)
                      {
                           if(cd[i][j]==0)
```

```
                        {
                            cd[i][j]=((cd[i][k]) * (cd[k][j]));
                        }
                    }
                }
            }
}
int Mostrar_centro(int m[DIM][DIM])
{
   int i,j,x,vetor[DIM];
   i=0;
   j=0;
      while(i!=7)
         {
            vetor[i]=0;
            i++;
         }
   i=0;
      while(j!=7)
         {
               while(i!=7)
                  {
                        if((m[i][j]) > vetor[j])
                           {
                              vetor[j]=m[i][j];
                           }
```

```
                i++;
            }
        j++;
        i=0;
      }
  x=99;
  i=0;
    while(i!=7)
      {
          if(vetor[i] < x)
            {
              x=vetor[i];
            }
        i++;
      }
  return(x+1);
}
void Montar_menu(int Grafo[DIM][DIM], int M_c[DIM][DIM], int M_p[DIM][DIM], int Conectividade[DIM][DIM])
{
  do{
    system("CLS");
    printf("1- Mostrar a matriz de adjacencia do grafo.\n");
    printf("2- Mostrar a matriz de menores custos do grafo.\n");
    printf("3- Mostrar a matriz de conectividade do grafo.\n");
    printf("4- Mostrar o caminho\n");
    printf("5- Sair");
```

```
printf("\nDigite a sua op: ");
scanf("%d",&op);
}while((op<1) || (op>7));
if(op==1)
{
    Mostrar(Grafo);
    system("PAUSE");
    Montar_menu(Grafo,M_c,M_p,Conectividade);
}
if(op==2)
{
    Mostrar(M_c);
    system("PAUSE");
    Montar_menu(Grafo,M_c,M_p,Conectividade);
}
if(op==3)
{
    Mostrar(Conectividade);
    system("PAUSE");
    Montar_menu(Grafo,M_c,M_p,Conectividade);
}
if(op==4)
{
    printf("Digite o vertice de origem: ");
    scanf("%d",&ori);
    while((ori<=0)||(ori>7))
```

```
        {
            printf("\nVertice invalido. \nDigite novamente.\n");
            printf("\nDigite o vertice de origem: ");
            scanf("%d",&ori);
        }
        printf("Digite o vertice de destino: ");
        scanf("%d",&des);
        while((des<=0)||(des>7))
        {
            printf("\nVertice invalido. \nDigite novamente.\n");
            printf("Digite o vertice de destino: ");
            scanf("%d\n\n",&des);
        }
        printf("%d => ",ori);
        Mostrar_caminho(M_p,ori-1,des-1);
        printf("%d\n\n",des);
        system("PAUSE");
        Montar_menu(Grafo,M_c,M_p,Conectividade);
    }
    if(op==5)
      {
        exit(1);
      }
}
void Inicializar_grafo()
{
```

```
M_conectar(Grafo,1,2,1);
M_conectar(Grafo,1,5,1);
M_conectar(Grafo,1,7,9);
M_conectar(Grafo,2,3,1);
M_conectar(Grafo,2,4,4);
M_conectar(Grafo,2,6,5);
M_conectar(Grafo,3,4,2);
M_conectar(Grafo,4,7,3);
M_conectar(Grafo,5,4,10);
M_conectar(Grafo,5,6,2);
M_conectar(Grafo,6,3,8);
M_conectar(Grafo,6,7,3);
M_conectar(Grafo,7,2,6);
M_conectar(Grafo1,1,2,1);
M_conectar(Grafo1,1,5,1);
M_conectar(Grafo1,1,7,1);
M_conectar(Grafo1,2,3,1);
M_conectar(Grafo1,2,4,1);
M_conectar(Grafo1,2,6,1);
M_conectar(Grafo1,3,4,1);
M_conectar(Grafo1,4,7,1);
M_conectar(Grafo1,5,4,1);
M_conectar(Grafo1,5,6,1);
M_conectar(Grafo1,6,3,1);
M_conectar(Grafo1,6,7,1);
M_conectar(Grafo1,7,2,1);
```

POR DENTRO DO TEMA

```
}
int main ()
{
   system("CLS");
   CarreGrafoar(Grafo,0);
   CarreGrafoar(M_c,0);
   CarreGrafoar(M_p,-1);
   CarreGrafoar(Grafo1,0);
   CarreGrafoar(Conectividade,0);
   Inicializar_grafo();
   M_custo(Grafo,M_c,M_p);
   M_conectividade(Grafo1,Conectividade);
   Montar_menu(Grafo,M_c,M_p,Conectividade);
}
```

ACOMPANHE NA WEB

Grafos I: Conceitos e Aplicações

- Introdução aos grafos: definição, terminologia, algumas propriedades e exemplos de aplicações de grafos.

Disponível em: <http://wiki.icmc.usp.br/images/5/59/Grafos_I.pdf> Acesso em: 12 ago. 2014.

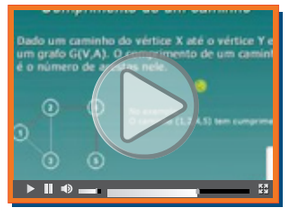
ACOMPANHENAWEB

Algoritmos em Grafos

- Suponha que existam seis sistemas computacionais (A, B, C, D, E e F) interconectados entre si. Esta informação pode ser representada por um diagrama chamado de grafo.

Disponível em: <http://homepages.dcc.ufmg.br/~loureiro/alg/052/pa2e_cap7.pdf> Acesso em: 12 ago. 2014.

Grafos - Estrutura de Dados 2



- Grafos é um tipo abstrato de estrutura de dados em que, dado um número de objetos, há um conjunto de conexões entre pares destes objetos.

Disponível em: <http://www.youtube.com/watch?v=_uk1E-h63Kk> Acesso em: 12 ago. 2014.

Tempo: 5:18.

AGORAÉASUAVEZ

Instruções:

Agora, chegou a sua vez de exercitar seu aprendizado. A seguir, você encontrará algumas questões de múltipla escolha e dissertativas. Leia cuidadosamente os enunciados e atente-se para o que está sendo pedido.

Questão 1

Dada uma árvore B de ordem 35, qual é o número máximo de chaves que podem ser armazenadas por nó folha e o número mínimo de chaves que um nó pode armazenar, exceto o nó raiz? Fundamente sua resposta.

AGORA É A SUA VEZ

Questão 2

Dentre as teorias que envolvem o estudo dos grafos, uma diz que o número de arcos que incidem em um nó determina o seu grau. Pode-se observar também que:

a) O número de arcos que um nó tem como cabeça representa seu grau de entrada.

b) O número de arcos que um nó tem como terminação de seta representa seu grau de entrada.

c) O número de arcos que um nó tem como cabeça representa seu grau de desvio padrão.

d) O número de arcos que um nó tem como terminação de seta representa seu grau de desvio padrão.

e) Nenhuma das alternativas é verdadeira.

Questão 3

Algumas estruturas de dados e técnicas de programação podem ser utilizadas nas implementações de soluções para problemas que envolvem as teorias dos grafos. Assinale a alternativa que corresponde a uma estrutura que é amplamente utilizada para implementar grafos:

a) Vetor unidimensional.

b) Matriz unidimensional.

c) Matriz de adjacência.

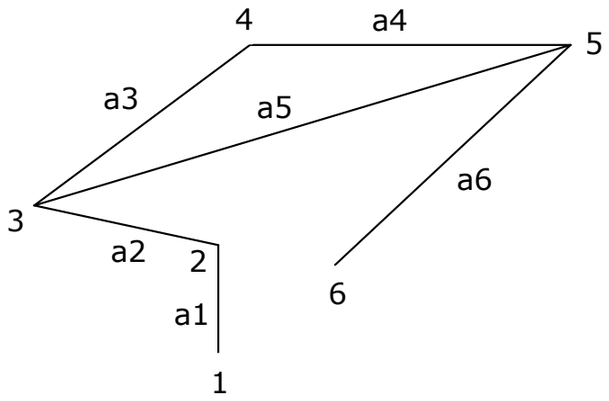
d) Vetor.

e) Nenhuma das alternativas é verdadeira.

AGORAÉASUAVEZ

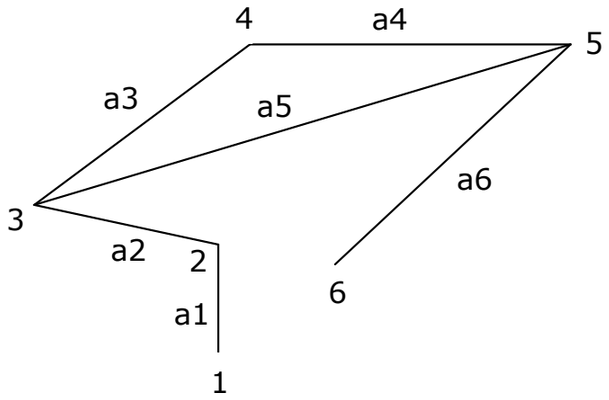
Questão 4

Observe o grafo a seguir. É correto afirmar que existem dois caminhos para percorrer do nó 3 para o nó 6? Se a resposta for sim, descreva os caminhos.



Questão 5

Observe o grafo a seguir. Existe algum ciclo no grafo? Se a resposta for sim, descreva os caminhos percorridos pelos ciclos.



FINALIZANDO

Os conceitos e exemplos práticos em C apresentados neste tema deixaram bem claro que os grafos são muito versáteis na solução de problemas do cotidiano.

Diferentemente de outros ramos da matemática, que muitas vezes apresentam trabalhos com base somente teórica, a teoria dos grafos é focada em problemas práticos do cotidiano. Seu embasamento está direcionado ao estudo de objetos combinatórios (grafos), que representam modelos para problemas em diversos ramos da matemática, computação, engenharia, indústria, entre outros.

Foram apresentados conceitos básicos sobre grafos e seu uso em C com alguns exemplos. Ao finalizar, tivemos alguns exercícios para reforçar o conteúdo estudado referente ao estudo dos grafos.

REFERÊNCIAS

CAMPELLO, Ricardo J. G. B. Grafos I: Conceitos e Aplicações. *Estruturas de Dados*. Disponível em: <http://wiki.icmc.usp.br/images/5/59/Grafos_I.pdf>. Acesso em: 28 jun. 2014.

LUOREIRO, Antonio Alfredo Ferreira. *Algoritmos e Estruturas de Dados II*: Algoritmos em Grafos. UFMG. Disponível em: <http://homepages.dcc.ufmg.br/~loureiro/alg/052/pa2e_cap7.pdf>. Acesso em: 28 jun. 2014.

SILVA, Luciano da; SILVEIRA, Artur Rafael da. Grafos: Estrutura de Dados 2. *Vídeo/YouTube*. Disponível em: <http://www.youtube.com/watch?v=_uk1E-h63Kk>. Acesso em: 28 jun. 2014.

TENENBAUM, Aaron M.; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. *Estruturas de Dados usando C*. São Paulo: Makron Books, 1995.

GLOSSÁRIO

Estrutura de dados: na computação, estrutura de dados é uma forma específica de organização e armazenamento de dados, para que sejam utilizados de forma eficaz. As estruturas de dados e seus algoritmos são muito utilizados na Ciência da Computação, em diversas áreas do conhecimento e com as mais diferentes finalidades na solução de problemas computacionais.

Aresta: na geometria, o encontro de duas faces resulta em linhas chamadas de arestas. A aresta também é chamada de “reta”.

Vértice: é o ponto comum a duas ou mais retas, também definido como ponto de ligação entre as arestas. Os vértices são geralmente representados por um ponto.

GABARITO

Questão 1

Resposta: Uma árvore B de ordem 35 pode ter no máximo 34 chaves por nó folha e armazenar no mínimo **17** chaves por nó.

Por definição, em uma árvore B de ordem m, o número máximo de chaves em um nó folha é m-1 e o número mínimo de chaves em um nó é dado por $\lceil m / 2 \rceil - 1$.

Questão 2

Resposta: Alternativa A.

Uma definição simples de grafo seria a seguinte: um conjunto de vértices (ou nós) e um conjunto de arestas (ou arcos), sendo que cada aresta em um grafo é representada por um par de vértices. Um grafo G=(V,A) consiste em:

GABARITO

1. Conjunto finito de vértices V. Tais elementos de V são chamados vértices do grafo G.

2. Conjunto finito A de pares não ordenados de V. Tais elementos de A são chamados arestas do grafo G.

Dizemos que um nó x qualquer incide no arco y somente se x representar no par ordenado de nós que forma y um dos dois nós. Neste caso, podemos dizer que y incide em x. O número de arcos que incidem em um nó determina o seu grau. Pode-se observar também que:

3. O número de arcos que um nó tem como cabeça representa seu grau de entrada.

4. O número de arcos que um nó tem como terminação de seta representa seu grau de saída.

Conforme pode ser observado nas definições anteriormente descritas, referentes aos conceitos de grafos, o item 3 em negrito corresponde à resposta da questão.

Questão 3

Resposta: Alternativa C.

A implementação de grafos em linguagens de programação exige estruturas de dados multidimensionais para representá-los. Isso significa que é necessário utilizar uma estrutura de dados que permita o armazenamento das informações de forma multidimensional. No caso da matriz, isso é possível através da estrutura de indexação por linhas e colunas, que permite a implementação de uma estrutura multidimensional. Tal estrutura aplicada na representação de grafos é chamada de matriz de adjacência.

Questão 4

Resposta: Sim. Existem dois caminhos do nó 3 para o nó 6.

1º caminho: 3 – a5 – 5 – a6 – 6.

2º caminho: 3 – a3 – 4 – a4 – 5 – a6 – 6.

Questão 5

Resposta: Sim. Existe apenas um ciclo: 3 – a3 – 4 – a4 – 5 – a5 – 3.


Anhanguera