

Descrição

Visualizar envios

Avaliação II - parte 2

Disponível a partir de: sexta, 21 mai 2021, 06:35

Data de entrega: sexta, 21 mai 2021, 12:00

Número máximo de arquivos: 1

Tipo de trabalho: Trabalho individual

Dado o pseudo-código

DFS(G)

1. **para** cada vértice $u \in V$ **faça**
2. $cor[u] \leftarrow$ branco; $pred[u] \leftarrow$ NULO;
3. $tempo \leftarrow 0$;
4. **para** cada vértice $u \in V$ **faça**
5. **se** $cor[u] =$ branco **então** DFS-AUX(u)

DFS-AUX(u) ▷ u acaba de ser descoberto

1. $cor[u] \leftarrow$ cinza;
2. $tempo \leftarrow tempo + 1$; $d[u] \leftarrow tempo$;
3. **para** $v \in Adj[u]$ **faça** ▷ explora aresta (u, v)
4. **se** $cor[v] =$ branco **então**
5. $pred[v] \leftarrow u$; DFS-AUX(v);
6. $cor[u] \leftarrow$ preto; ▷ u foi explorado
7. $tempo \leftarrow tempo + 1$; $f[u] \leftarrow tempo$;

implemente as funções DFS_aux e DFS_recursivo segundo o pseudo-código acima.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  #define true 1
6  #define false 0
7  typedef int bool;
8  typedef int TIPOPESO;
9  /* Vértices de grafos são representados por objetos do tipo vertex. */
10 #define maxV 1024
11 #define BRANCO 0
12 #define CINZA 1
13 #define PRETO 2
14
15 static int pa[1000];
16 static int cnt, d[maxV], f[maxV], dist[maxV], cor[maxV], pred[maxV];
17 int tempo = 0;
18
19
20
21 typedef struct adjacencia{

```

```

22     int vertice;
23     TIPOPESO peso;
24     struct adjacencia *prox;
25 } ADJACENCIA;
26
27 typedef struct vertice{
28     /* Dados armazenados vao aqui */
29     ADJACENCIA *cab;
30 } VERTICE;
31
32 typedef struct grafo {
33     int vertices;
34     int arestas;
35     VERTICE *adj;
36 } GRAFO;
37
38 typedef struct no{
39     int u;
40     ADJACENCIA *p;
41 }NO;
42
43 NO *pilha;
44 int fim;
45
46 /* Criando um grafo */
47 GRAFO *criarGrafo(int v){
48     GRAFO *g = (GRAFO *) malloc(sizeof(GRAFO));
49
50     g->vertices = v;
51     g->arestas = 0;
52     g->adj = (VERTICE *) malloc(v*sizeof(VERTICE));
53     int i;
54
55     for (i=0; i<v; i++)
56         g->adj[i].cab = NULL;
57
58     return g;
59 }
60
61 ADJACENCIA *criaAdj(int v,int peso){
62     ADJACENCIA *temp = (ADJACENCIA *) malloc(sizeof(ADJACENCIA));
63     temp->vertice = v;
64     temp->peso = peso;
65     temp->prox = NULL;
66     return (temp);
67 }
68
69 bool criaAresta(GRAFO *gr, int vi, int vf, TIPOPESO p){
70     if (!gr)
71         return(false);
72     if((vf<0) || (vf >= gr->vertices))
73         return(false);
74     if((vi<0) || (vf >= gr->vertices))
75         return(false);
76
77     ADJACENCIA *novo = criaAdj(vf,p);
78
79     novo->prox = gr->adj[vi].cab;
80     gr->adj[vi].cab = novo;
81
82     ADJACENCIA *novo2 = criaAdj(vi,p);
83
84     novo2->prox = gr->adj[vf].cab;
85     gr->adj[vf].cab = novo2;
86
87     gr->arestas++;

```

```

88     return (true);
89 }
90
91 void imprime(GRAFO *gr){
92     printf("Vertices: %d. Arestas: %d, \n", gr->vertices,gr->arestas);
93
94     int i;
95     for(i=0;i<gr->vertices; i++){
96         printf("v%d: ",i);
97         ADJACENCIA *ad = gr->adj[i].cab;
98         while(ad){
99             printf("v%d(%d) ", ad->vertice,ad->peso);
100             ad = ad->prox;
101         }
102
103         printf("\n");
104     }
105 }
106
107 void STACKinit(int maxN){
108     pilha = (NO*) malloc (maxN*sizeof(NO));
109     fim = 0;
110
111
112 }
113
114
115 int STACKempty(){
116     return fim == 0;
117 }
118
119 void STACKput(int item, ADJACENCIA *px){
120     pilha[fim].u = item;
121     pilha[fim].p = px;
122     fim++;
123
124 }
125
126 NO STACKget(){
127     return pilha[--fim];
128 }
129
130 void STACKfree(){
131     free(pilha);
132
133 }
134 void DFS_aux(GRAFO *G,int u);
135 void DFS_recursivo(GRAFO *G, int raiz);
136
137 void imprimeTree(GRAFO *gr){
138     for(int v=0; v < gr->vertices; v++){
139         printf("(%d,%d)\n",pred[v],v);
140     }
141
142 }
143
144 int main(){
145     int a,b,w;
146     GRAFO *gr = criarGrafo(12);
147     for(int i=0; i < 12; i++){
148         scanf("%i",&a);
149         scanf("%i",&b);
150         scanf("%i",&w);
151         criaAresta(gr,a,b,w);
152     }
153

```

```

154     DFS_recursivo(gr,0);
155     imprimeTree(gr);
156
157     return 0;
158 }
159
160 /* implemente as funções DFS_aux e DFS_recursivo */
161
162 void DFS_aux(GRAFO *G,int u){
163     // seu código aqui
164 }
165 void DFS_recursivo(GRAFO *G, int raiz){
166     // seu código aqui
167 }

```



PARTICIPANTES



VPL



Participantes



MENSAGENS



ROBSON DA SILVA LOPES

1

Mensagens



CALENDÁRIO



Fevereiro 2022						
Seg (Segunda- feira)	Ter (Terça- feira)	Qua (Quarta- feira)	Qui (Quinta- feira)	Sex (Sexta- feira)	Sáb (Sábado)	Dom (Domingo)
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

CHAVE DE EVENTOS

- 👁 Ocultar eventos globais
- 👁 Ocultar eventos de curso
- 👁 Ocultar eventos de grupo
- 👁 Ocultar eventos de usuário



PRÓXIMOS EVENTOS



Não há nenhum evento próximo

[Ir ao calendário...](#)

[Novo evento...](#)



ÚLTIMOS AVISOS



(Nenhum aviso publicado.)

Página inicial do AVA

Portal de Sistemas Integrados

Páginas

Meus cursos

20172 - BMD - FILOSOFIA E METODOLOGIA CIENTÍFICA

20172 - CC - MECÂNICA

20172 - CC - Matemática Discreta

20172 - CC - Estrutura de Dados I

20172 - CC - ESTATISTICA GERAL

20172 - CC - CÁLCULO I

20201 - CC1 - Empreendedorismo

20201 - CC1 - Inteligência Artificial

20201 - CC - Introdução à Computação Gráfica

20201 - CC - Programação IV

■ Mais...

Cursos

20201 - CC - Estrutura de Dados II

Participantes

 Notas

Geral

REVISÃO C

Tabelas e Funções Hash

Árvores AVL

Arvores B e B+

Árvores Vermelho e Preto

Matriz Esparsa


Grafos e digrafos

Algoritmos de buscas

Outros algoritmos em Grafos

AVALIAÇÕES

 AVALIAÇÃO I

 Avaliação II - parte 1

 **Avaliação II - parte 2**

■ Descrição

■ Visualizar envios

Fone: +55 (65) 3615-8028

Contato: ces@ufmt.br