

```

1  #include <stdio.h>
2  #define MAX_Length 20
3
4  typedef struct
5  {
6      /* data */
7      int u, v; // u = dinh 1, v = dinh 2
8  }edge;
9
10 typedef struct
11 {
12     /* data */
13     int n, m; // n = so dinh, m = so canh
14     edge data[MAX_Length]; // e = tap hop cac canh
15 }graph;
16
17 typedef struct
18 {
19     int data[MAX_Length];
20     int size;
21 }list;
22
23 void makeNull(list *L){
24     L->size = 0;
25 }
26 int elementAt(list *L, int x){
27     return L->data[x];
28 }
29 void pushList(list *L, int x){
30     L->data[L->size] = x;
31     L->size++;
32 }
33
34 void initGraph(graph *G, int n){
35     G->n = n;
36     G->m = 0;
37     printf("Do thi G duoc khoi tao voi so dinh n = %d va so canh m = %d\n", G->n, G->m);
38 }
39
40 void addEdge(graph *G, int u, int v){
41     int i;
42     for(i = 0; i < G->m; i++){
43         if ((G->data[i].u == u && G->data[i].v == v) || (G->data[i].u == v && G->data[i].v == u))
44         {
45             printf("Canh da co trong do thi!");
46             return;
47         }
48     }
49     G->data[G->m].u = u;
50     G->data[G->m].v = v;
51     G->m++;
52     printf("Do thi da them vao canh %d %d\n", G->data[G->m-1].u, G->data[G->m-1].v);
53 }
54 int adjacent(graph *G, int u, int v){
55     int i;
56     for(i = 0; i <= G->m; i++){
57         if ((G->data[i].u == u && G->data[i].v == v) || (G->data[i].v == u && G->data[i].u == v))
58         {
59             return 1;
60         }
61     }
62     return 0;
63 }
64
65 void degree(graph *G, int x){
66     int i, count = 0;
67     for(i = 0; i <= G->m; i++){
68         if (adjacent(G, x, i))
69         {
70             count++;
71         }
72     }
73     printf("Bac cua canh %d la: %d\n", x, count);
74 }

```

```

76 list neighbors(graph *G, int x){
77     list listVertex;
78     makeNull(&listVertex);
79     int i;
80     for(i = 1; i<=G->n; i++){
81         if (adjacent(G, x, i) == 1)
82             {
83                 pushList(&listVertex, i);
84             }
85     }
86     return listVertex;
87 }
88
89 // Stack
90
91 typedef struct {
92     int data[100];
93     int size;
94 }Stack;
95
96 void make_null_stack(Stack *S) {
97     S->size = 0;
98 }
99
100 void push(Stack *S, int x) {
101     S->data[S->size] = x;
102     S->size++;
103 }
104
105 void pop(Stack *S) {
106     S->size--;
107 }
108
109 int top(Stack S) {
110     return S.data[S.size-1];
111 }
112
113 int empty(Stack S) {
114     return S.size == 0;
115 }
116
117 // DUYET THEO CHIEU SAU
118
119 void depth_first_search(graph *G) {
120     printf("\nDuyet DFS:\n");
121     Stack L;
122     int mark[100];
123     make_null_stack(&L);
124
125     int i;
126     for (i = 1; i<= G->n; i++)
127         mark[i] = 0;
128
129
130     for(i = 1; i<=G->n; i++) {
131         if(mark[i] == 0) {
132             push(&L, i);
133             while(!empty(L)) {
134                 int x = top(L);
135                 pop(&L);
136                 if(mark[x] != 0) continue;
137                 printf("Duyet %d\n",x );
138                 mark[x] =1;
139                 list ls = neighbors(G, x);
140                 for (i = 0; i< ls.size; i++) {
141                     int e = elementAt(&ls, i);
142                     push(&L, e);
143                 }
144             }
145             printf("\n");
146         }
147     }
148
149 }
150 }
151

```

```

152 // Queue
153 typedef struct {
154     int data[100];
155     int front, rear;
156 }Queue;
157
158 void make_null_queue(Queue* Q) {
159     Q->front = 0;
160     Q->rear = -1;
161 }
162
163 void pushQ(Queue *Q, int x) {
164     Q->rear++;
165     Q->data[Q->rear] = x;
166 }
167
168 int topQ(Queue Q) {
169     return Q.data[Q.front];
170 }
171
172 void popQ(Queue *Q) {
173     Q->front++;
174 }
175
176 int emptyQ(Queue Q) {
177     return Q.front > Q.rear;
178 }
179
180 void breath_first_search(graph *G) {
181     printf("\nDuyet BFS:\n");
182     Queue L;
183     int mark[100];
184     make_null_queue(&L);
185
186     int i, j;
187     for(i = 1; i<= G->n; i++)
188         mark[i] = 0;
189
190     for( j = 1; j<=G->n; j++) {
191         if(mark[j] == 0) {
192             pushQ(&L, j);
193             printf("Duyet %d\n", j);
194             mark[j] = 1;
195             while(!emptyQ(L)) {
196                 int x = topQ(L);
197                 popQ(&L);
198                 list ls = neighbors(G, x);
199                 for ( i = 0; i< ls.size; i++) {
200                     int e = elementAt(&ls, i);
201                     if(mark[e] == 0) {
202                         printf("Duyet %d\n", e);
203                         mark[e] = 1;
204                         pushQ(&L, e);
205                     }
206                 }
207             }
208         }
209     }
210     printf("\n");
211 }
212
213 }
214
215 }

```

```

216 int main(int argc, char const *argv[])
217 {
218     freopen("dothi.txt", "r", stdin);
219     int n, m, u, v, i, j;
220     scanf("%d%d", &n, &m);
221     graph G;
222     initGraph(&G, n);
223     for(i = 1; i<=m; i++){
224         scanf("%d%d", &u, &v);
225         addEdge(&G, u, v);
226     }
227     for(i = 1; i<=n; i++){
228         degree(&G, i);
229     }
230     for(i = 1; i<=n; i++){
231         list l = neighbors(&G, i);
232         printf("neighbor(%d): ", i);
233         for(j = 0; j<l.size; j++){
234             printf("%d ", elementAt(&l, j));
235         }
236         printf("\n");
237     }
238     depth_first_search(&G);
239     breath_first_search(&G);
240     return 0;
241 }

```

