**THỰC HÀNH BUỔI 1**

1. **BT1. DSC: hàm init\_graph**

|  |
| --- |
| void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  } |

1. **BT2. DSC: hàm add\_edge cơ bản**

|  |
| --- |
| void add\_edge(Graph \*G, int u, int v){  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  } |

1. **BT3. DSC: hàm add\_edge nâng cao**

|  |  |
| --- | --- |
| Q1.  void add\_edge(Graph \*G, int u, int v){  if(u<1 || v>G->n || v<1 || u>G->n) return;  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  } | Q2.  void add\_edge(Graph \*G, int u, int v){  for(int i = 0; i < G->m; i++){  if(G->edges[i].u == u && G->edges[i].v == v){  return;  }  }  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  } |
| Q3.  void add\_edge(Graph \*G, int u, int v){  if(u == v) return;  for(int i=0;i<G->m;i++){  if((G->edges[i].u == u && G->edges[i].v == v) || (G->edges[i].u == v && G->edges[i].v == u))  return;  }  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  } | |

1. **BT4. DSC: hàm adjacent**

|  |
| --- |
| Q1.  int adjacent(Graph \*G, int u, int v){  int e;  for(e=0;e<G->m;e++)  if((G->edges[e].u == u && G->edges[e].v == v) || (G->edges[e].u == v && G->edges[e].v == u))  return 1;  return 0;  } |
| Q2.  int adjacent(Graph \*G, int u, int v){  int e;  for(e=0;e<G->m;e++)  if(G->edges[e].u == u && G->edges[e].v == v)  return 1;  return 0;  } |

1. **BT5. DSC: tổng hợp**

|  |
| --- |
| Q1.  #define MAX\_N 50  typedef struct{  int u, v;  }Edge;  typedef struct{  Edge edges[MAX\_N];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  }  int degree(Graph \*G, int u){  int deg = 0;  for(int e = 0; e < G->m; e++){  if(G->edges[e].u == u)  deg++;  if(G->edges[e].v == u)  deg++;  }  return deg;  } |
| Q2.  #include<stdio.h>  #define MAX 500  typedef struct{  int u, v;  }Edge;  typedef struct{  Edge edges[MAX];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  }  int degree(Graph \*G, int u){  int deg = 0;  for(int e = 0; e < G->m; e++){  if(G->edges[e].u == u)  deg++;  if(G->edges[e].v == u)  deg++;  }  return deg;  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1; u <= G.n; u++)  printf("deg(%d) = %d\n", u, degree(&G, u));  } |
| Q3.  #include<stdio.h>  #define MAX 500  typedef struct{  int u, v;  }Edge;  typedef struct{  Edge edges[MAX];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  }  int degree(Graph \*G, int u){  int deg = 0;  for(int e = 0; e < G->m; e++){  if(G->edges[e].u == u)  deg++;  if(G->edges[e].v == u)  deg++;  }  return deg;  }  int main(){  Graph G;  int n, m, u, v, e;  freopen("dt.txt", "r", stdin);  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1; u <= G.n; u++)  printf("deg(%d) = %d\n", u, degree(&G, u));  } |

1. **BT6. DSC: liệt kê đỉnh kề:**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX 500  typedef struct{  int u, v;  }Edge;  typedef struct{  Edge edges[MAX];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  int e;  for(e = 0; e < G->m; e++)  if((G->edges[e].u == u && G->edges[e].v == v) || (G->edges[e].u == v && G->edges[e].v == u))  return 1;  return 0;  }  void neighbours(Graph \*G, int u){  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v))  printf("%d ", v);  printf("\n");  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1; u <= G.n; u++){  printf("neighbours(%d) = ", u);  neighbours(&G, u);  }  } |
| Q2.  #include<stdio.h>  #define MAX 500  typedef struct{  int u, v;  }Edge;  typedef struct{  Edge edges[MAX];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  int e;  for(e=0;e<G->m;e++)  if(G->edges[e].u == u && G->edges[e].v == v)  return 1;  return 0;  }  void neighbours(Graph \*G, int u){  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v))  printf("%d ", v);  printf("\n");  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1; u <= G.n; u++){  printf("neighbours(%d) = ", u);  neighbours(&G, u);  }  } |

1. **BT7. MTK: hàm init\_graph và add\_edge**

|  |
| --- |
| Q1.  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  G->m++;  } |
| Q2.  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  } |
| Q3.  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  G->m++;  } |

1. **BT8. MTK: hàm add\_edge**

|  |
| --- |
| Q1.  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  if(u != v)  G->A[v][u] = 1;  G->m++;  } |
| Q2.  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->m++;  } |
| Q3.  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  } |
| Q4.  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  G->m++;  } |

1. **BT9. MTK: In ma trận kề**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  printf("Ma tran ke: \n");  for(u = 1; u <= G.n; u++){  for(v = 1; v <= G.n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  } |
| Q2.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->m++;  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  printf("Ma tran ke: \n");  for(u = 1; u <= G.n; u++){  for(v = 1; v <= G.n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  } |
| Q3.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  printf("Ma tran ke: \n");  for(u = 1; u <= G.n; u++){  for(v = 1; v <= G.n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  } |
| Q4.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N];  int n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  // if(u != v)  // G->A[v][u] += 1;  G->m++;  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  printf("Ma tran ke: \n");  for(u = 1; u <= G.n; u++){  for(v = 1; v <= G.n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  } |

1. **BT10. MTK: hàm degree**

|  |
| --- |
| Q1.  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  int degree(Graph \*G, int u){  int deg = 0;  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v))  deg += G->A[u][v];  return deg + G->A[u][u];  } |
| Q2.  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  int degree(Graph \*G, int u){  int deg = 0;  for(int v = 1; v <= G->n; v++)  deg += G->A[v][u] + G->A[u][v];  return deg;  } |
| Q3.  int indegree(Graph \*G, int u){  int deg = 0;  for(int v = 1; v <= G->n; v++)  deg += G->A[v][u];  return deg;  } |
| Q4.  int outdegree(Graph \*G, int u){  int deg = 0;  for(int v = 1; v <= G->n; v++)  deg += G->A[u][v];  return deg;  } |

1. **BT11. MTK: liệt kê đỉnh kề**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  void neighbours(Graph \*G, int u){  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v))  printf("%d ", v);  printf("\n");  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1; u <= G.n; u++){  printf("neighbours(%d) = ", u);  neighbours(&G, u);  }  return 0;  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] += 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  void neighbours(Graph \*G, int u){  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v))  printf("%d ", v);  printf("\n");  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1; u <= G.n; u++){  printf("neighbours(%d) = ", u);  neighbours(&G, u);  }  return 0;  } |

1. **BT 13-14-15. Chuyển đổi các phương pháp biểu diễn**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 200  #define MAX\_M 500  typedef struct{  int n, m;  int A[MAX\_N][MAX\_N];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u, v;  for(u=1;u<=n;u++)  for(v=1;v<=n;v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  G->m++;  }  void neighbours(Graph \*G, int u){  int v;  for(v=1;v<=G->n;v++)  if(G->A[u][v] != 0)  printf("%d ",v);  }  int main(){  Graph G;  int n, u, v, k;  scanf("%d", &n);  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  scanf("%d",&G.A[u][v]);  }  }  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  if(u<=v){  for(k=1;k<=G.A[u][v];k++)  printf("%d %d\n", u, v);  }  }  }  return 0;  } |
| Q2.  #include<stdio.h>  #define MAX\_N 200  #define MAX\_M 500  typedef struct{  int n, m;  int A[MAX\_N][MAX\_N];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u, v;  for(u=1;u<=n;u++)  for(v=1;v<=n;v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->m++;  }  void neighbours(Graph \*G, int u){  int v;  for(v=1;v<=G->n;v++)  if(G->A[u][v] != 0)  printf("%d ",v);  }  int main(){  Graph G;  int n, u, v, k;  scanf("%d", &n);  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  scanf("%d",&G.A[u][v]);  }  }  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  for(k=1;k<=G.A[u][v];k++)  printf("%d %d\n", u, v);  }  }  return 0;  } |
| Q3.  #include<stdio.h>  #define MAX\_N 200  #define MAX\_M 500  typedef struct{  int n, m;  int A[MAX\_N][MAX\_N];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u, v;  for(u=1;u<=n;u++)  for(v=1;v<=n;v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  G->m++;  }  void neighbours(Graph \*G, int u){  int v;  for(v=1;v<=G->n;v++)  if(G->A[u][v] != 0)  printf("%d ",v);  }  int main(){  Graph G;  int n, u, v, k;  scanf("%d", &n);  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  scanf("%d",&G.A[u][v]);  }  }  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  for(k=1;k<=G.A[u][v];k++)  printf("%d ", v);  }  printf("0\n");  }  return 0;  } |
| Q4.  #include<stdio.h>  #define MAX\_N 200  #define MAX\_M 500  typedef struct{  int n, m;  int A[MAX\_N][MAX\_N];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u, v;  for(u=1;u<=n;u++)  for(v=1;v<=n;v++)  G->A[u][v] = 0;  }  void add\_edge(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->m++;  }  void neighbours(Graph \*G, int u){  int v;  for(v=1;v<=G->n;v++)  if(G->A[u][v] != 0)  printf("%d ",v);  }  int main(){  Graph G;  int n, u, v, k;  scanf("%d", &n);  // init\_graph(&G, n);  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  scanf("%d",&G.A[u][v]);  }  }  for(u=1;u<=n;u++){  for(v=1;v<=n;v++){  for(k=1;k<=G.A[u][v];k++)  printf("%d ", v);  }  printf("0\n");  }  return 0;  } |
| Q5.  #include <stdio.h>  #define MAX\_N 100  typedef struct {  int n;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;    for (int u = 1; u <= n; u++)  for (int v = 1; v <= n; v++)  pG->A[u][v] = 0;  }  void add\_edge(Graph \*pG, int u, int v) {  pG->A[u][v] += 1;  }  //Hàm main()  int main() {  Graph G;  int n;  scanf("%d", &n);  G.n = n;  for (int u = 1; u <= n; u++) {  int v;  while (1) {  scanf("%d", &v);  if (v == 0)  break;  add\_edge(&G, u, v);  }  }  for (int u = 1; u <= n; u++) {  for (int v = 1; v <= n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  return 0;  } |
| Q6.  #include <stdio.h>  #define MAX\_N 100  typedef struct {  int n;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;    for (int u = 1; u <= n; u++)  for (int v = 1; v <= n; v++)  pG->A[u][v] = 0;  }  void add\_edge(Graph \*pG, int u, int v) {  pG->A[u][v] += 1;  }  //Hàm main()  int main() {  Graph G;  int n;  scanf("%d", &n);  G.n = n;  for (int u = 1; u <= n; u++) {  int v;  while (1) {  scanf("%d", &v);  if (v == 0)  break;  add\_edge(&G, u, v);  }  }  for (int u = 1; u <= n; u++) {  for (int v = 1; v <= n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  return 0;  } |

\*BT bổ sung:

1. **Bài 1 - MTĐ - C: hàm add\_edge**

|  |
| --- |
| void add\_edge(Graph \*G, int e, int x, int y){  G->A[x][e] = 1;  G->A[y][e] = 1;  } |

1. **Bài 2 - MTĐ - C: hàm neighbours trả về List**

|  |
| --- |
| List neighbours(Graph\* G, int x) {  List L;  make\_null(&L);  int e, y;  for (y = 1; y <= G->n; y++) {  if (x == y) continue;  for (e = 1; e <= G->m; e++)  if (G->A[x][e] > 0 && G->A[y][e] > 0) {  push\_back(&L, y);  break;  }  }  return L;  } |

1. **Bài 3 - MTK: hàm deg**

|  |
| --- |
| int deg(Graph \*G, int u){  int deg\_u = 0;  for(int v = 1; v <= G->n; v++)  deg\_u += G->A[u][v];  return deg\_u + G->A[u][u];  } |

1. **Bài 4 - Ứng dụng: Tìm đỉnh có bậc lớn nhất**

|  |
| --- |
| #include<stdio.h>  #define MAX\_VERTICES 100  #define MAX\_EDGES 500  #define MAX\_ELEMENTS 100  typedef struct{  int n, m;  int A[MAX\_VERTICES][MAX\_EDGES];  } Graph;  void init\_graph(Graph \*G, int n, int m){  int i, j;  G->n = n;  G->m = m;  for (i = 1; i <= n; i++)  for (j = 1; j <= m; j++)  G->A[i][j] = 0;  }  void add\_edge(Graph \*G, int e, int x, int y){  G->A[x][e] = 1;  G->A[y][e] = 1;  }  int degree(Graph \*G, int x){  int e, deg = 0;  for (e = 0; e < G->m; e++)  if (G->A[x][e] == 1)  deg++;  return deg;  }  int adjacent(Graph \*G, int x, int y){  int e;  for (e = 1; e <= G->m; e++)  if (G->A[x][e] == 1 && G->A[y][e] == 1)  return 1;  return 0;  }  typedef struct {  int data[MAX\_ELEMENTS];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->data[i-1];  }  int count\_list(List \*L){  return L->size;  }  List neighbors(Graph \*G, int x){  int y;  List L;  make\_null(&L);  for (y = 1; y <= G->n; y++)  if (adjacent(G, x, y) && y != x)  push\_back(&L, y);  return L;    }  void count(Graph \*G){  int dinh, i, max = 0;  for (i = 1; i <= G->n; i++){  if(degree(G, i) > max){  max = degree(G, i);  dinh = i;  }  }  printf("%d %d\n", dinh, max);  }  int main() {  Graph G;  int n, m, u, v, e;  scanf("%d%d", &n, &m);  init\_graph(&G, n, m);  for (e = 1; e <= m; e++){  scanf("%d%d", &u, &v);  add\_edge(&G, e, u, v);  }  count(&G);  return 0;  } |

1. **Bài 5 - Danh sách cung**

|  |
| --- |
| Q1.  void add\_edge(Graph \*G, int x, int y){  G->edges[G->m].x = x;  G->edges[G->m].y = y;  G->m++;  } |
| Q2.  void add\_edge(Graph \*G, int x, int y){  if(x<1 || y<1 || x>G->n || y>G->n) return;  else{  G->edges[G->m].x = x;  G->edges[G->m].y = y;  G->m++;  }  } |

1. **Bài 6 - Đọc đồ thị từ tập tin**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N];  int n, m;  }Graph;  void kt(Graph \*G, int n){  G->n = n;  G->m = 0;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = 0;  }  void themcung(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  G->m++;  }  int main(){  Graph G;  int n, m, u, v;  freopen("dt1.txt", "r", stdin);  scanf("%d%d",&n,&m);  kt(&G, n);  for(int e = 0; e < m; e++){  scanf("%d%d",&u,&v);  themcung(&G, u, v);  }  for(u = 1; u <= G.n; u++){  for(v = 1; v <= G.n; v++)  printf("%d ", G.A[u][v]);  printf("\n");  }  } |