**THỰC HÀNH BUỔI 2**

1. **BT1. BFS - Duyệt theo chiều rộng từ 1 đỉnh**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define MAX\_SIZE 100  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->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int hangdoi[MAX\_SIZE];  int front, rear;  }Queue;  void make\_null(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->rear++;  Q->hangdoi[Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int empty(Queue \*Q){  return Q->front > Q->rear;  }  int mark[MAX\_N];  void bfs(Graph \*G, int s){  Queue Q;  make\_null(&Q);  enQueue(&Q, s);  while(!empty(&Q)){  int u = front(&Q);  deQueue(&Q);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G, u, v))  enQueue(&Q, v);  }  }  }  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);  }  bfs(&G, 1);  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define MAX\_SIZE 100  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->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int hangdoi[MAX\_SIZE];  int front, rear;  }Queue;  void make\_null(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->rear++;  Q->hangdoi[Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int empty(Queue \*Q){  return Q->front > Q->rear;  }  int mark[MAX\_N];  void bfs(Graph \*G, int s){  Queue Q;  make\_null(&Q);  enQueue(&Q, s);  while(!empty(&Q)){  int u = front(&Q);  deQueue(&Q);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G, u, v))  enQueue(&Q, v);  }  }  }  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);  }  bfs(&G, 1);  } |
| Q3.  #include<stdio.h>  #define MAX\_N 50  #define MAX\_SIZE 100  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->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int hangdoi[MAX\_SIZE];  int front, rear;  }Queue;  void make\_null(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->rear++;  Q->hangdoi[Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int empty(Queue \*Q){  return Q->front > Q->rear;  }  int mark[MAX\_N];  void bfs(Graph \*G, int s){  Queue Q;  make\_null(&Q);  enQueue(&Q, s);  while(!empty(&Q)){  int u = front(&Q);  deQueue(&Q);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G, u, v))  enQueue(&Q, v);  }  }  }  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);  }  int s;  scanf("%d",&s);  bfs(&G, s);  } |
| Q4.  #include<stdio.h>  #define MAX\_N 50  #define MAX\_SIZE 100  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->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int hangdoi[MAX\_SIZE];  int front, rear;  }Queue;  void make\_null(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->rear++;  Q->hangdoi[Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int empty(Queue \*Q){  return Q->front > Q->rear;  }  int mark[MAX\_N];  void bfs(Graph \*G, int s){  Queue Q;  make\_null(&Q);  enQueue(&Q, s);  while(!empty(&Q)){  int u = front(&Q);  deQueue(&Q);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G, u, v))  enQueue(&Q, v);  }  }  }  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);  }  int s;  scanf("%d",&s);  bfs(&G, s);  } |

1. **BT2. BFS - Duyệt theo chiều rộng TOÀN BỘ đồ thị**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define MAX\_SIZE 100  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->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int hangdoi[MAX\_SIZE];  int front, rear;  }Queue;  void make\_null(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->rear++;  Q->hangdoi[Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int empty(Queue \*Q){  return Q->front > Q->rear;  }  int mark[MAX\_N];  void bfs(Graph \*G, int s){  Queue Q;  make\_null(&Q);  enQueue(&Q, s);  while(!empty(&Q)){  int u = front(&Q);  deQueue(&Q);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G, u, v))  enQueue(&Q, v);  }  }  }  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++)  mark[u] = 0;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  bfs(&G, u);  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define MAX\_SIZE 100  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 adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int hangdoi[MAX\_SIZE];  int front, rear;  }Queue;  void make\_null(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->rear++;  Q->hangdoi[Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int empty(Queue \*Q){  return Q->front > Q->rear;  }  int mark[MAX\_N];  void bfs(Graph \*G, int s){  Queue Q;  make\_null(&Q);  enQueue(&Q, s);  while(!empty(&Q)){  int u = front(&Q);  deQueue(&Q);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G, u, v))  enQueue(&Q, v);  }  }  }  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++)  mark[u] = 0;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  bfs(&G, u);  } |

1. **BT3. DFS - Duyệt theo chiều sâu từ 1 đỉnh**

|  |
| --- |
| 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;  G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int nganxep[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->size++] = u;  }  int top(Stack \*S){  return S->nganxep[S->size - 1];  }  void pop(Stack \*S){  S->size--;  }  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int s){  Stack S;  make\_null(&S);  push\_back(&S, s);  for(int i = 1; i <= G->n; i++)  mark[i] = 0;  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  for(int v = G->n; v >= 1; v--){  if(adjacent(G, u, v))  push\_back(&S, v);  }  }  }  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++)  mark[u] = 0;  dfs(&G, 1);  } |
| 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->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int nganxep[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->size++] = u;  }  int top(Stack \*S){  return S->nganxep[S->size - 1];  }  void pop(Stack \*S){  S->size--;  }  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int s){  Stack S;  make\_null(&S);  push\_back(&S, s);  for(int i = 1; i <= G->n; i++)  mark[i] = 0;  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  for(int v = G->n; v >= 1; v--){  if(adjacent(G, u, v))  push\_back(&S, v);  }  }  }  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++)  mark[u] = 0;  dfs(&G, 1);  } |
| 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;  G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int nganxep[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->size++] = u;  }  int top(Stack \*S){  return S->nganxep[S->size - 1];  }  void pop(Stack \*S){  S->size--;  }  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int s){  Stack S;  make\_null(&S);  push\_back(&S, s);  for(int i = 1; i <= G->n; i++)  mark[i] = 0;  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  for(int v = G->n; v >= 1; v--){  if(adjacent(G, u, v))  push\_back(&S, v);  }  }  }  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++)  mark[u] = 0;  int s;  scanf("%d",&s);  dfs(&G, s);  } |
| 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;  // G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int nganxep[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->size++] = u;  }  int top(Stack \*S){  return S->nganxep[S->size - 1];  }  void pop(Stack \*S){  S->size--;  }  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int s){  Stack S;  make\_null(&S);  push\_back(&S, s);  for(int i = 1; i <= G->n; i++)  mark[i] = 0;  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] == 1)  continue;  printf("%d\n", u);  mark[u] = 1;  for(int v = G->n; v >= 1; v--){  if(adjacent(G, u, v))  push\_back(&S, v);  }  }  }  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++)  mark[u] = 0;  int s;  scanf("%d",&s);  dfs(&G, s);  } |

1. **BT4. DFS - Duyệt theo chiều sâu TOÀN BỘ đồ thị**

|  |
| --- |
| 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;  G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int nganxep[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->size++] = u;  }  int top(Stack \*S){  return S->nganxep[S->size - 1];  }  void pop(Stack \*S){  S->size--;  }  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int s){  Stack S;  make\_null(&S);  push\_back(&S, s);  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] != 0)  continue;  printf("%d\n", u);  mark[u] = 1;  for(int v = G->n; v >= 1; v--){  if(adjacent(G, u, v))  push\_back(&S, v);  }  }  }  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++)  mark[u] = 0;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  dfs(&G, u);  } |
| 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 adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  typedef struct{  int nganxep[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->size++] = u;  }  int top(Stack \*S){  return S->nganxep[S->size - 1];  }  void pop(Stack \*S){  S->size--;  }  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int s){  Stack S;  make\_null(&S);  push\_back(&S, s);  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] != 0)  continue;  printf("%d\n", u);  mark[u] = 1;  for(int v = G->n; v >= 1; v--){  if(adjacent(G, u, v))  push\_back(&S, v);  }  }  }  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++)  mark[u] = 0;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  dfs(&G, u);  } |

1. **BT6. Xây dựng cây duyệt đồ thị**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc Hang doi  typedef struct{  int data[MAX\_Element];  int front, rear;  }Queue;  //Khoi tao hang doi rong  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  //Them mot phan tu vao trong hang doi  void push\_Queue(Queue \*Q, int x){  Q->rear++;  Q->data[Q->rear] = x;  }  //Kiem tra hang doi co rong hay khong  int empty\_Queue(Queue \*Q){  return (Q->front > Q->rear);  }  //Lay mot phan tu o dau hang doi  int top(Queue \*Q){  return Q->data[Q->front];  }  void pop(Queue \*Q){  Q->front++;  }  List breath\_first\_search(Graph \*G, int x, int parent[]){  Queue Q;  make\_null\_Queue(&Q);  int mark[MAX\_Vertices];  int i;  for(i=1;i<=G->n;i++)  mark[i] = 0;  push\_Queue(&Q, x);  parent[x] = -1;  List L\_bfs;  make\_null(&L\_bfs);  while(!empty\_Queue(&Q)){  int u = top(&Q);  pop(&Q);  if(mark[u] == 1)  continue;  push\_back(&L\_bfs, u);  mark[u] = 1;  List L;  make\_null(&L);  L = neighbours(G, u);  int v;  for(i=1;i<=L.size;i++){  v = element\_at(&L, i);  if(mark[v] == 0){  push\_Queue(&Q, v);  if(parent[v] == -1)  parent[v] = u;  }  }  }  return L\_bfs;  }  int main(){  Graph G;  // freopen("BFS\_data.txt", "r", stdin);  int n,m,i,j;  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int x, y, e;  for(e=1;e<=m;e++){  scanf("%d%d",&x,&y);  add\_edge(&G, x , y);  }  int mark\_bfs[MAX\_Vertices], parent[MAX\_Vertices];  for(i=1;i<=G.n;i++){  mark\_bfs[i] = 0;  parent[i] = -1;  }  for(i=1;i<=G.n;i++){  if(mark\_bfs[i] == 0){  List L = breath\_first\_search(&G, i, parent);  for(j=1;j<=L.size;j++){  int v = element\_at(&L, j);  // printf("%d\n", v);  mark\_bfs[v] = 1;  }  }  printf("%d %d\n", i ,parent[i]);  }  } |
| Q2.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  // G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc Hang doi  typedef struct{  int data[MAX\_Element];  int front, rear;  }Queue;  //Khoi tao hang doi rong  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  //Them mot phan tu vao trong hang doi  void push\_Queue(Queue \*Q, int x){  Q->rear++;  Q->data[Q->rear] = x;  }  //Kiem tra hang doi co rong hay khong  int empty\_Queue(Queue \*Q){  return (Q->front > Q->rear);  }  //Lay mot phan tu o dau hang doi  int top(Queue \*Q){  return Q->data[Q->front];  }  void pop(Queue \*Q){  Q->front++;  }  List breath\_first\_search(Graph \*G, int x, int parent[]){  Queue Q;  make\_null\_Queue(&Q);  int mark[MAX\_Vertices];  int i;  for(i=1;i<=G->n;i++)  mark[i] = 0;  push\_Queue(&Q, x);  parent[x] = -1;  List L\_bfs;  make\_null(&L\_bfs);  while(!empty\_Queue(&Q)){  int u = top(&Q);  pop(&Q);  if(mark[u] == 1)  continue;  push\_back(&L\_bfs, u);  mark[u] = 1;  List L;  make\_null(&L);  L = neighbours(G, u);  int v;  for(i=1;i<=L.size;i++){  v = element\_at(&L, i);  if(mark[v] == 0){  push\_Queue(&Q, v);  if(parent[v] == -1)  parent[v] = u;  }  }  }  return L\_bfs;  }  int main(){  Graph G;  // freopen("BFS\_data.txt", "r", stdin);  int n,m,i,j;  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int x, y, e;  for(e=1;e<=m;e++){  scanf("%d%d",&x,&y);  add\_edge(&G, x , y);  }  int mark\_bfs[MAX\_Vertices], parent[MAX\_Vertices];  for(i=1;i<=G.n;i++){  mark\_bfs[i] = 0;  parent[i] = -1;  }  for(i=1;i<=G.n;i++){  if(mark\_bfs[i] == 0){  List L = breath\_first\_search(&G, i, parent);  for(j=1;j<=L.size;j++){  int v = element\_at(&L, j);  // printf("%d\n", v);  mark\_bfs[v] = 1;  }  }  printf("%d %d\n", i ,parent[i]);  }  } |
| Q3.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], 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 them(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  int mark[MAX\_N], parent[MAX\_N];  void dfs(Graph \*G, int u, int p){  if(mark[u] == 1) return;  mark[u] = 1;  parent[u] = p;  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v) && !mark[v]){  dfs(G, v, u);  }  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++){  mark[u] = 0;  parent[v] = -1;  }  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  dfs(&G, u, -1);  for(u = 1; u <= G.n; u++)  printf("%d %d\n", u, parent[u]);  } |
| Q4.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], 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 them(Graph \*G, int u, int v){  G->A[u][v] = 1;  // G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  int mark[MAX\_N], parent[MAX\_N];  void dfs(Graph \*G, int u, int p){  if(mark[u] == 1) return;  mark[u] = 1;  parent[u] = p;  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v) && !mark[v]){  dfs(G, v, u);  }  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++){  mark[u] = 0;  parent[v] = -1;  }  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  dfs(&G, u, -1);  for(u = 1; u <= G.n; u++)  printf("%d %d\n", u, parent[u]);  } |

1. **BT7. Kiểm tra đồ thị vô hướng liên thông**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], 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 them(Graph \*G, int u, int v){  G->A[u][v] = 1;  G->A[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] > 0;  }  int mark[MAX\_N];  void dfs(Graph \*G, int u){  if(mark[u] == 1) return;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(adjacent(G, u, v) && !mark[v]){  dfs(G, v);  }  }  int connected(Graph \*G){  int u;  for(u = 1; u <= G->n; u++)  mark[u] = 0;  dfs(G, 1);  for(u = 1; u <= G->n; u++)  if(mark[u] == 0)  return 0;  return 1;  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++){  mark[u] = 0;  }  if(connected(&G))  printf("CONNECTED");  else printf("DISCONNECTED");  } |

1. **BT8. Bộ phận liên thông**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }DT;  void kt(DT \*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 them(DT \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int mark[MAX\_N];  void dfs(DT \*G, int u){  if(mark[u] == 1) return;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] != 0 && !mark[v])  dfs(G, v);  }  int main(){  DT G;  int u, v, n, m, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++)  mark[u] = 0;  int cnt = 0;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0){  dfs(&G, u);  cnt++;  }  printf("%d", cnt);  } |
| Q2.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }DT;  void kt(DT \*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 them(DT \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int mark[MAX\_N], cnt = 0;  void dfs(DT \*G, int u){  if(mark[u] == 1) return;  cnt++;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] != 0 && !mark[v])  dfs(G, v);  }  int main(){  DT G;  int u, v, n, m, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++)  mark[u] = 0;  dfs(&G, 1);  printf("%d", cnt);  } |
| Q3.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }DT;  void kt(DT \*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 them(DT \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int mark[MAX\_N], cnt = 0;  void dfs(DT \*G, int u){  if(mark[u] == 1) return;  cnt++;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] != 0 && !mark[v])  dfs(G, v);  }  int main(){  DT G;  int u, v, n, m, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++)  mark[u] = 0;  int s;  scanf("%d",&s);  dfs(&G, s);  printf("%d", cnt);  } |
| Q4.  #include<stdio.h>  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }DT;  void kt(DT \*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 them(DT \*G, int u, int v){  G->A[u][v] += 1;  if(u != v)  G->A[v][u] += 1;  G->m++;  }  int mark[MAX\_N], cnt = 0, max\_cnt = 0;  void dfs(DT \*G, int u){  if(mark[u] == 1) return;  cnt++;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] != 0 && !mark[v])  dfs(G, v);  if(cnt > max\_cnt)  max\_cnt = cnt;  }  int main(){  DT G;  int u, v, n, m, e;  scanf("%d%d",&n,&m);  kt(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  them(&G, u, v);  }  for(u = 1; u <= G.n; u++)  mark[u] = 0;  max\_cnt = 0;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0){  cnt = 0;  dfs(&G, u);  }  printf("%d", max\_cnt);  } |

1. **BT9. Ứng dụng liên thông**

|  |
| --- |
| Q1.  #include <stdio.h>  /\* Khai báo CTDL Graph\*/  #define MAX\_N 100  typedef struct {  int n, m;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;  pG->m = 0;  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;  if (u != v)  pG->A[v][u] += 1;    pG->m++;  }  int adjacent(Graph \*pG, int u, int v) {  return pG->A[u][v] > 0;  }  //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt  int mark[MAX\_N];  void DFS(Graph \*pG, int u) {  //1. Đánh dấu u đã duyệt  //printf("Duyet %d\n", u); //Làm gì đó trên u  mark[u] = 1; //Đánh dấu nó đã duyệt  //2. Xét các đỉnh kề của u  for (int v = 1; v <= pG->n; v++)  if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt  DFS(pG, v); //gọi đệ quy duyệt nó  }  int connected(Graph \*pG) {  int u;  for (u = 1; u <= pG->n; u++)  mark[u] = 0;  DFS(pG, 1);  for (u = 1; u <= pG->n; u++)  if (mark[u] == 0)  return 0;  return 1;  }  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++) {  mark[u] = 0;  }  if (connected(&G))  printf("YES\n");  else  printf("NO\n");  return 0;  } |
| Q2.  #include <stdio.h>  /\* Khai báo CTDL Graph\*/  #define MAX\_N 100  typedef struct {  int n, m;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;  pG->m = 0;  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;  if (u != v)  pG->A[v][u] += 1;    pG->m++;  }  int adjacent(Graph \*pG, int u, int v) {  return pG->A[u][v] > 0;  }  //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt  int mark[MAX\_N];  void DFS(Graph \*pG, int u) {  //1. Đánh dấu u đã duyệt  //printf("Duyet %d\n", u); //Làm gì đó trên u  mark[u] = 1; //Đánh dấu nó đã duyệt  //2. Xét các đỉnh kề của u  for (int v = 1; v <= pG->n; v++)  if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt  DFS(pG, v); //gọi đệ quy duyệt nó  }  int connected(Graph \*pG) {  int u;  for (u = 1; u <= pG->n; u++)  mark[u] = 0;  DFS(pG, 1);  for (u = 1; u <= pG->n; u++)  if (mark[u] == 0)  return 0;  return 1;  }  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++) {  mark[u] = 0;  }  if (connected(&G))  printf("DUOC\n");  else  printf("KHONG\n");  return 0;  } |

1. **BT10. Kiểm tra đồ thị chứa chu trình**

|  |
| --- |
| Q1.  #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(color[v] == WHITE)  DFS(G, v);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u);  if(has\_circle == 0) printf("NO CIRCLE");  else printf("CIRCLED");  } |
| Q2.  #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(color[v] == WHITE)  DFS(G, v);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u);  if(has\_circle == 0){  printf("-1");  }  } |
| Q3.  #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u, int p){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(v == p)  continue;  if(color[v] == WHITE)  DFS(G, v, u);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u, -1);  if(has\_circle == 0) printf("NO CIRCLE");  else printf("CIRCLED");  } |
| Q4.  #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u, int p){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(v == p)  continue;  if(color[v] == WHITE)  DFS(G, v, u);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u, -1);  if(has\_circle == 0) printf("-1");  } |

1. **BT11. Ứng dụng kiểm tra chu trình**

|  |
| --- |
| Q1.  #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(color[v] == WHITE)  DFS(G, v);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u);  if(has\_circle == 0) printf("OK");  else printf("CIRCULAR REFERENCE");  } |
| Q2.  #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(color[v] == WHITE)  DFS(G, v);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u);  if(has\_circle == 0) printf("YES");  else printf("NO");  } |

1. **BT12. Kiểm tra đồ thị phân đôi**

|  |
| --- |
| Q1.  #include<stdio.h>  #define khong\_mau 0  #define xanh 1  #define do 2  #define MAX\_N 40  typedef struct{  int A[MAX\_N][MAX\_N];  int dinh, cung;  }DoThi;  void khoitao(DoThi \*D, int dinh){  D->dinh = dinh;  D->cung = 0;  int c1, c2;  for(c1=1;c1<=D->dinh;c1++)  for(c2=1;c2<=D->dinh;c2++)  D->A[c1][c2] = 0;  }  void them\_cung(DoThi \*D, int c1, int c2){  D->A[c1][c2] = 1;  D->A[c2][c1] = 1;  D->cung++;  }  int hohang(DoThi \*D, int c1, int c2){  return D->A[c1][c2] != 0;  }  int mau[MAX\_N], dungdo;  void tomau(DoThi \*D, int c1, int m){  mau[c1] = m;  int c2;  for(c2=1;c2<=D->dinh;c2++)  if(hohang(D, c1, c2)){  if(mau[c2] == khong\_mau)  tomau(D, c2, 3-m);  else if(mau[c2] == mau[c1])  dungdo = 1;  }  }  int main(){  DoThi D;  int dinh, cung, c1, c2, e;  scanf("%d%d",&dinh,&cung);  khoitao(&D, dinh);  for(e=0;e<cung;e++){  scanf("%d%d",&c1,&c2);  them\_cung(&D,c1,c2);  }  dungdo = 0;  for(c1=1;c1<=D.dinh;c1++)  mau[c1] = khong\_mau;  for(c1=1;c1<=D.dinh;c1++)  if(mau[c1] == khong\_mau)  tomau(&D, c1, xanh);  if(dungdo == 1) printf("NO");  else printf("YES");  } |
| Q2.  #include <stdio.h>  #define MAX\_N 100  #define NO\_COLOR 0  #define BLUE 1  #define RED 2  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++;  }  int adjacent(Graph \*G, int u, int v) {  return G->A[u][v] > 0;  }  int color[MAX\_N];  int conflict = 0;  void colorize(Graph \*G, int u, int c){  color[u]=c;  for(int v=1;v<=G->n;v++){  if(adjacent(G,u,v)){  if(color[v]==NO\_COLOR){  colorize(G,v,3-c);  }  else if(color[v]==color[u]){  conflict=1;  }  }  }  }  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);  }  int i;  for(i=1;i<=G.n;i++)  {  color[u]=NO\_COLOR;  }    conflict=0;  colorize(&G,1,BLUE);  if(conflict==0){  for(i=1;i<=G.n;i++){  if(color[i]==BLUE){  printf("%d ",i);  }  }  printf("\n");  for(i=1;i<=G.n;i++){  if(color[i]==RED){  printf("%d ",i);  }  }  }  else{  printf("IMPOSSIBLE");  }  return 0;  } |

1. **BT13. Tính liên thông mạnh**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 100  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 adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  typedef struct{  int data[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push(Stack \*S, int u){  S->data[S->size] = u;  S->size++;  }  int top(Stack \*S){  return S->data[S->size-1];  }  void pop(Stack \*S){  S->size--;  }  int num[MAX\_N], min\_num[MAX\_N], k, on\_stack[MAX\_N];  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  make\_null(&S);  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1;v<=G->n;v++)  if(adjacent(G, u, v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min( min\_num[u], min\_num[v] );  }  else if(on\_stack[v]) min\_num[u] = min(min\_num[u], num[v]);  }  if(num[u] == min\_num[u]){  // printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);  int w;  do{  w = top(&S); pop(&S);  on\_stack[u] = 0;  }while(w != u);  }  }  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);  }  Stack S;  make\_null(&S);  for(u=1;u<=G.n;u++)  num[u] = -1;  k = 1;  for(u=1;u<=G.n;u++){  if(num[u] == -1)  SCC(&G, u);  printf("%d %d\n", num[u], min\_num[u]);  }  return 0;  } |
| Q2.  #include<stdio.h>  #define MAX\_N 100  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 adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  typedef struct{  int data[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push(Stack \*S, int u){  S->data[S->size] = u;  S->size++;  }  int top(Stack \*S){  return S->data[S->size-1];  }  void pop(Stack \*S){  S->size--;  }  int num[MAX\_N], min\_num[MAX\_N], k, on\_stack[MAX\_N], dem;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  make\_null(&S);  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1;v<=G->n;v++)  if(adjacent(G, u, v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min( min\_num[u], min\_num[v] );  }  else if(on\_stack[v]) min\_num[u] = min(min\_num[u], num[v]);  }  if(num[u] == min\_num[u]){  // printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);  int w;  dem++;  do{  w = top(&S); pop(&S);  on\_stack[u] = 0;  }while(w != u);  }  }  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);  }  Stack S;  make\_null(&S);  for(u=1;u<=G.n;u++)  num[u] = -1;  k = 1;  dem = 0;  for(u=1;u<=G.n;u++){  if(num[u] == -1){  SCC(&G, u);    }  // printf("%d %d\n", num[u], min\_num[u]); (bài 1)  }  if(dem == 1) printf("STRONG CONNECTED");  else printf("DISCONNECTED");  return 0;  } |
| Q3.  #include<stdio.h>  #define MAX\_N 100  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 adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  typedef struct{  int data[MAX\_N];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push(Stack \*S, int u){  S->data[S->size] = u;  S->size++;  }  int top(Stack \*S){  return S->data[S->size-1];  }  void pop(Stack \*S){  S->size--;  }  int num[MAX\_N], min\_num[MAX\_N], k, on\_stack[MAX\_N], dem, cnt;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  make\_null(&S);  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1;v<=G->n;v++)  if(adjacent(G, u, v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min( min\_num[u], min\_num[v] );  }  else if(on\_stack[v]) min\_num[u] = min(min\_num[u], num[v]);  }  if(num[u] == min\_num[u]){  // printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);  int w;  dem++;  do{  cnt++;  w = top(&S); pop(&S);  on\_stack[u] = 0;  }while(w != u);  }  }  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);  }  Stack S;  make\_null(&S);  for(u=1;u<=G.n;u++)  num[u] = -1;  k = 1;  // dem = 0; (bài 2)  cnt = 0;  for(u=1;u<=G.n;u++){  if(num[u] == -1){  SCC(&G, u);    }  // printf("%d %d\n", num[u], min\_num[u]); (bài 1)  }  // if(dem == 1) printf("STRONG CONNECTED");  // else printf("DISCONNECTED"); (bài 2)  printf("%d", cnt);  return 0;  } |
| Q4.  #include <stdio.h>  /\* Khai báo CTDL Graph\*/  #define MAX\_N 100  typedef struct {  int n, m;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;  pG->m = 0;  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;  //if (u != v)  // pG->A[v][u] += 1;    if (pG->A[u][v] > 1)  printf("da cung (%d, %d)\n", u, v);  if (u == v)  printf("khuyen %d\n", u);      pG->m++;  }  int adjacent(Graph \*pG, int u, int v) {  return pG->A[u][v] > 0;  }  #define MAX\_SIZE 100  typedef int ElementType;  typedef struct {  ElementType data[MAX\_SIZE];  int top\_idx;  } Stack;  /\* Hàm khởi tạo ngăn xếp rỗng \*/  void make\_null\_stack(Stack \*pS) {  pS->top\_idx = -1;  }  /\* Hàm thêm phần tử u vào đỉnh ngăn xếp \*/  void push(Stack \*pS, ElementType u) {  pS->top\_idx++;  pS->data[pS->top\_idx] = u;  }  /\* Hàm xem phần tử trên đỉnh ngăn xếp \*/  ElementType top(Stack \*pS) {  return pS->data[pS->top\_idx];  }  /\* Hàm xoá bỏ phần tử trên đỉnh ngăn xếp \*/  void pop(Stack \*pS) {  pS->top\_idx--;  }  /\* Hàm kiểm tra ngăn xếp rỗng \*/  int empty(Stack \*pS) {  return pS->top\_idx == -1;  }  int min(int a, int b) {  return a < b ? a : b;  }  int num[MAX\_N], min\_num[MAX\_N];  int k;  Stack S;  int on\_stack[MAX\_N];  int max\_cnt;  //Duyệt đồ thị bắt đầu từ đỉnh u  void SCC(Graph \*pG, int u) {  //1. Đánh số u, đưa u vào ngăn xếp S  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  //2. Xét các đỉnh kề của u  for (int v = 1; v <= pG->n; v++) {  if (adjacent(pG, u, v)) {  if (num[v] < 0) {  SCC(pG, v);  min\_num[u] = min(min\_num[u], min\_num[v]);  } else if (on\_stack[v])  min\_num[u] = min(min\_num[u], num[v]);  }  }  //3. Kiểm tra u có phải là đỉnh khớp  if (num[u] == min\_num[u]) {  //printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);  int nb\_cnt = 0;  int w;  do { //Lấy các đỉnh trong S ra cho đến khi gặp u  w = top(&S);  pop(&S);  on\_stack[w] = 0;  //printf("Lay %d.\n", w);  nb\_cnt++;  } while (w != u);  if (nb\_cnt > max\_cnt)  max\_cnt = nb\_cnt;  }  }  int main() {  //1. Khai báo đồ thị G  Graph G;  //2. Đọc dữ liệu và dựng đồ thị  int n, m, u, v;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (int e = 0; e < m; e++) {  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  }  for (int u = 1; u <= G.n; u++)  num[u] = -1;  //3. Duyệt toàn bộ đồ thị để kiểm tra chu trình  k = 1; //1b. Tất cả đều chưa duyệt  make\_null\_stack(&S); //1c. Làm rỗng ngăn xếp  //2. Duyệt toàn bộ đồ thị để tìm BPLT mạnh  max\_cnt = 0;  for (int u = 1; u <= G.n; u++)  if (num[u] == -1) //u chưa duyệt  SCC(&G, u); //duyệt nó    printf("%d\n", max\_cnt);  return 0;  } |

1. **BT14. Ứng dụng liên thông mạnh.**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 100  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;  }  typedef struct{  int ngxep[MAX\_N], top\_idx;  }Stack;  void make\_null(Stack \*S){  S->top\_idx = 0;  }  void push(Stack \*S, int u){  S->ngxep[S->top\_idx] = u;  S->top\_idx++;  }  int top(Stack \*S){  return S->ngxep[S->top\_idx - 1];  }  void pop(Stack \*S){  S->top\_idx--;  }  int num[MAX\_N], min\_num[MAX\_N], on\_stack[MAX\_N], k, dem = 0, p;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1; v<=G->n;v++){  if(adjacent(G,u,v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min(min\_num[u], min\_num[v]);  }  else if(on\_stack[v])  min\_num[u] = min(min\_num[u], num[v]);  }  }  if(num[u] == min\_num[u]){  int w;  dem++;  do{  w = top(&S); pop(&S);  on\_stack[w] = 0;  }while(w != u);  }  }  int main(){  Graph G;  Stack S;  int u, v, n, m;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(int e = 0; e<m; e++){  scanf("%d%d%d",&u,&v,&p);  if(p == 1)  add\_edge(&G,u,v);  if(p == 2){  add\_edge(&G, u, v);  add\_edge(&G, v ,u);  }  }  for(u = 1;u <= G.n; u++){  num[u] = -1;  on\_stack[u] = 0;  }  k = 1;  make\_null(&S);  for(u = 1;u <= G.n; u++)  if(num[u] == -1)  SCC(&G, u);  if(dem == 1) printf("OKIE");  else printf("NO");  } |
| Q2.  #include<stdio.h>  #define MAX\_N 100  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;  }  typedef struct{  int ngxep[MAX\_N], top\_idx;  }Stack;  void make\_null(Stack \*S){  S->top\_idx = 0;  }  void push(Stack \*S, int u){  S->ngxep[S->top\_idx] = u;  S->top\_idx++;  }  int top(Stack \*S){  return S->ngxep[S->top\_idx - 1];  }  void pop(Stack \*S){  S->top\_idx--;  }  int num[MAX\_N], min\_num[MAX\_N], on\_stack[MAX\_N], k, dem;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1; v<=G->n;v++){  if(adjacent(G,u,v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min(min\_num[u], min\_num[v]);  }  else if(on\_stack[v])  min\_num[u] = min(min\_num[u], num[v]);  }  }  if(num[u] == min\_num[u]){  int w;  dem++;  do{  w = top(&S); pop(&S);  on\_stack[w] = 0;  }while(w != u);  }  }  int main(){  Graph G;  Stack S;  int u, v, n, m;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(int e = 0; e<m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for(u = 1;u <= G.n; u++){  num[u] = -1;  on\_stack[u] = 0;  }  k = 1;  dem = 0;  make\_null(&S);  for(u = 1;u <= G.n; u++)  if(num[u] == -1)  SCC(&G, u);  printf("%d ", dem);  } |

\*BT bổ sung:

1. **BS - Kiểm tra chu trình - đồ thị vô hướng**

|  |
| --- |
| #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u, int p){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(v == p)  continue;  if(color[v] == WHITE)  DFS(G, v, u);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u, -1);  if(has\_circle == 0){  printf("NO");  }  else printf("YES");  } |

1. **BS - Thuyền trưởng Haddock**

|  |
| --- |
| #include<stdio.h>  #define WHITE 0  #define GRAY 1  #define BLACK 2  #define MAX\_N 40  #define MAX\_Vertices 20  typedef struct{  int n,m;  int A[MAX\_Vertices][MAX\_Vertices];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u,v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int color[MAX\_N], has\_circle;  void DFS(Graph \*G, int u){  color[u] = GRAY;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G,u,v)){  if(color[v] == WHITE)  DFS(G, v);  else if(color[v] == GRAY)  has\_circle = 1;  }  color[u] = BLACK;  }  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++)  color[u] = WHITE;  has\_circle = 0;  for(u=1;u<=G.n;u++)  if(color[u] == WHITE)  DFS(&G, u);  if(has\_circle == 0){  printf("YES");  }  else printf("NO");  } |

1. **BS - Phân chia đội bóng**

|  |
| --- |
| #include <stdio.h>  #define MAX\_N 100  #define NO\_COLOR 0  #define BLUE 1  #define RED 2  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++;  }  int adjacent(Graph \*G, int u, int v) {  return G->A[u][v] > 0;  }  int color[MAX\_N];  int conflict = 0;  void colorize(Graph \*G, int u, int c){  color[u]=c;  for(int v=1;v<=G->n;v++){  if(adjacent(G,u,v)){  if(color[v]==NO\_COLOR){  colorize(G,v,3-c);  }  else if(color[v]==color[u]){  conflict=1;  }  }  }  }  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);  }  int i;  for(i=1;i<=G.n;i++)  {  color[u]=NO\_COLOR;  }    conflict=0;  colorize(&G,1,BLUE);  if(conflict==0){  for(i=1;i<=G.n;i++){  if(color[i]==BLUE){  printf("%d ",i);  }  }  printf("\n");  for(i=1;i<=G.n;i++){  if(color[i]==RED){  printf("%d ",i);  }  }  }  else{  printf("IMPOSSIBLE");  }  return 0;  } |

1. **BS - Kiểm tra tính liên thông mạnh - đếm số BPLT mạnh**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 100  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;  }  typedef struct{  int data[MAX\_N], top\_idx;  }Stack;  void make\_null(Stack \*S){  S->top\_idx = 0;  }  void push(Stack \*S, int u){  S->data[S->top\_idx] = u;  S->top\_idx++;  }  int top(Stack \*S){  return S->data[S->top\_idx - 1];  }  void pop(Stack \*S){  S->top\_idx--;  }  int num[MAX\_N], min\_num[MAX\_N], on\_stack[MAX\_N], k, dem = 0;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1; v<=G->n;v++){  if(adjacent(G,u,v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min(min\_num[u], min\_num[v]);  }  else if(on\_stack[u])  min\_num[u] = min(min\_num[u], num[v]);  }  }  if(num[u] == min\_num[u]){  dem++;  int w;  do{  w = top(&S); pop(&S);  on\_stack[w] = 0;  }while(w != u);  }  }  int main(){  Graph G;  int u, v, n, m;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(int e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  Stack S;  make\_null(&S);  k = 1;  for(u = 1; u<= G.n; u++){  num[u] = -1;  on\_stack[u] = 0;  }  dem = 0;  for(u = 1; u<= G.n; u++){  if(num[u] == -1)  SCC(&G, u);  }  if(dem == 1) printf("STRONG CONNECTED");  else printf("DISCONNECTED");  } |

|  |
| --- |
| Q2.  #include<stdio.h>  #define MAX\_N 100  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;  }  typedef struct{  int data[MAX\_N], top\_idx;  }Stack;  void make\_null(Stack \*S){  S->top\_idx = 0;  }  void push(Stack \*S, int u){  S->data[S->top\_idx] = u;  S->top\_idx++;  }  int top(Stack \*S){  return S->data[S->top\_idx - 1];  }  void pop(Stack \*S){  S->top\_idx--;  }  int num[MAX\_N], min\_num[MAX\_N], on\_stack[MAX\_N], k, dem = 0;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  num[u] = min\_num[u] = k; k++;  push(&S, u);  on\_stack[u] = 1;  for(int v = 1; v<=G->n;v++){  if(adjacent(G,u,v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min(min\_num[u], min\_num[v]);  }  else if(on\_stack[u])  min\_num[u] = min(min\_num[u], num[v]);  }  }  if(num[u] == min\_num[u]){  dem++;  int w;  do{  w = top(&S); pop(&S);  on\_stack[w] = 0;  }while(w != u);  }  }  int main(){  Graph G;  int u, v, n, m;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(int e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  Stack S;  make\_null(&S);  k = 1;  for(u = 1; u<= G.n; u++){  num[u] = -1;  on\_stack[u] = 0;  }  dem = 0;  for(u = 1; u<= G.n; u++){  if(num[u] == -1)  SCC(&G, u);  }  printf("%d",dem);  } |

1. **BS - Come and Go**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  #define MAX\_Size 400  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  int u, v;  for(u=1;u<=G->n;u++)  for(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++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  typedef struct{  int data[MAX\_Size];  int size;  }Stack;  void make\_null(Stack \*S){  S->size = 0;  }  void push(Stack \*S, int x){  S->data[S->size] = x;  S->size++;  }  int top(Stack \*S){  return S->data[S->size-1];  }  void pop(Stack \*S){  S->size--;  }  int num[MAX\_N], min\_num[MAX\_N], k, on\_stack[MAX\_N], dem = 0, count = 0;  int min(int a, int b){  return (a < b ? a : b);  }  void SCC(Graph \*G, int u){  Stack S;  num[u] = min\_num[u] = k;  k++;  push(&S, u);  on\_stack[u] = 1;  int v;  for(v=1;v<=G->n;v++){  if(adjacent(G, u, v)){  if(num[v] < 0){  SCC(G, v);  min\_num[u] = min(min\_num[u], min\_num[v]);  }  else if(on\_stack[v]) min\_num[u] = min(min\_num[u], num[v]);    }  }  if(num[u] == min\_num[u]){  // printf("Tim duoc BPLT manh, %d la dinh khop.\n", u);  dem++;  int w;  do{  w = top(&S);  count++;  pop(&S);  on\_stack[u] = 0;  }while(w != u);  }  }  int main(){  Graph G;  int n, m, u, v, e, p;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e=0;e<m;e++){  scanf("%d%d%d",&u,&v,&p);  if(p==1)  add\_edge(&G,u,v);  if(p==2){  add\_edge(&G,u,v);  add\_edge(&G,v,u);  }  }  Stack S;  make\_null(&S);  for(u=1;u<=G.n;u++){  num[u] = -1;  on\_stack[u] = 0;  }  k = 1;  for(u=1;u<=G.n;u++){  if(num[u] == -1){  SCC(&G, u);  }  }  if(dem == 1) printf("OKIE");  else printf("NO");  return 0;  } |

1. **BS - Duyệt đồ thị**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc Hang doi  typedef struct{  int data[MAX\_Element];  int front, rear;  }Queue;  //Khoi tao hang doi rong  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  //Them mot phan tu vao trong hang doi  void push\_Queue(Queue \*Q, int x){  Q->rear++;  Q->data[Q->rear] = x;  }  //Kiem tra hang doi co rong hay khong  int empty\_Queue(Queue \*Q){  return (Q->front > Q->rear);  }  //Lay mot phan tu o dau hang doi  int top(Queue \*Q){  return Q->data[Q->front];  }  void pop(Queue \*Q){  Q->front++;  }  List breath\_first\_search(Graph \*G, int x){  Queue Q;  make\_null\_Queue(&Q);  int mark[MAX\_Vertices];  int i;  for(i=1;i<=G->n;i++)  mark[i] = 0;  push\_Queue(&Q, x);  List L\_bfs;  make\_null(&L\_bfs);  while(!empty\_Queue(&Q)){  int u = top(&Q);  pop(&Q);  if(mark[u] == 1)  continue;  push\_back(&L\_bfs, u);  mark[u] = 1;  List L;  make\_null(&L);  L = neighbours(G, u);  int v;  for(i=1;i<=L.size;i++){  v = element\_at(&L, i);  if(mark[v] == 0)  push\_Queue(&Q, v);  }  }  return L\_bfs;  }  int main(){  Graph G;  // freopen("BFS\_data.txt", "r", stdin);  int n,m,i,j;  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int x, y, e;  for(e=1;e<=m;e++){  scanf("%d%d",&x,&y);  add\_edge(&G, x , y);  }  int mark\_bfs[MAX\_Vertices];  for(i=1;i<=G.n;i++)  mark\_bfs[i] = 0;  for(i=1;i<=G.n;i++){  if(mark\_bfs[i] == 0){  List L = breath\_first\_search(&G, i);  for(j=1;j<=L.size;j++){  int v = element\_at(&L, j);  printf("%d\n", v);  mark\_bfs[v] = 1;  }  }  }  return 0;  } |
| Q2.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc ngan xep  typedef struct{  int data[MAX\_Element];  int size;  }Stack;  //Khoi tao ngan xep rong  void make\_null\_stack(Stack \*S){  S->size = 0;  }  //Them mot phan tu vao trong ngan xep  void push\_stack(Stack \*S, int x){  S->data[S->size] = x;  S->size++;  }  //Lay mot phan tu trong Stack  int top(Stack \*S){  return S->data[S->size-1];  }  //Xoa mot phan tu thuoc Stack  void pop(Stack \*S){  S->size--;  }  //Kiem tra Stack co rong hay khong  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_Vertices]; //Danh dau xem danh do duoc duyet hay chua  void depth\_first\_search(Graph \*G, int u){  if(mark[u] == 1) return;  printf("%d\n",u);  mark[u] = 1;  List list = neighbours(G,u);  int i;  for(i=1;i<=list.size;i++){  int v = element\_at(&list, i);  depth\_first\_search(G,v);  }  }  int main(){  Graph G;  int n,m;  // freopen("DFS.txt", "r", stdin);  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int u,v,e;  for(e=1;e<=m;e++){  scanf("%d%d",&u,&v);  add\_edge(&G,u,v);  }  for(u=1;u<=G.n;u++)  mark[u] = 0;  for(u=1;u<=G.n;u++)  if(mark[u] == 0)  depth\_first\_search(&G, u);  return 0;  } |
| Q3.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc ngan xep  typedef struct{  int data[MAX\_Element];  int size;  }Stack;  //Khoi tao ngan xep rong  void make\_null\_stack(Stack \*S){  S->size = 0;  }  //Them mot phan tu vao trong ngan xep  void push\_stack(Stack \*S, int x){  S->data[S->size] = x;  S->size++;  }  //Lay mot phan tu trong Stack  int top(Stack \*S){  return S->data[S->size-1];  }  //Xoa mot phan tu thuoc Stack  void pop(Stack \*S){  S->size--;  }  //Kiem tra Stack co rong hay khong  int empty(Stack \*S){  return S->size == 0;  }  int mark[MAX\_Vertices]; //Danh dau xem danh do duoc duyet hay chua  void depth\_first\_search(Graph \*G, int x){  Stack S;  make\_null\_stack(&S);  push\_stack(&S, x);  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] != 0) continue;  printf("%d\n", u);  mark[u] = 1;  int v;  for(v=1;v<=G->n;v++)  if(adjacent(G, u, v))  push\_stack(&S,v);  }  }  int main(){  Graph G;  int n,m;  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int u,v,e;  for(e=1;e<=m;e++){  scanf("%d%d",&u,&v);  add\_edge(&G,u,v);  }  for(u=1;u<=G.n;u++)  mark[u] = 0;  for(u=1;u<=G.n;u++)  if(mark[u] == 0)  depth\_first\_search(&G, u);  return 0;  } |

1. **BS - Duyệt đồ thị & Dựng cây duyệt đồ thị**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc Hang doi  typedef struct{  int data[MAX\_Element];  int front, rear;  }Queue;  //Khoi tao hang doi rong  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  //Them mot phan tu vao trong hang doi  void push\_Queue(Queue \*Q, int x){  Q->rear++;  Q->data[Q->rear] = x;  }  //Kiem tra hang doi co rong hay khong  int empty\_Queue(Queue \*Q){  return (Q->front > Q->rear);  }  //Lay mot phan tu o dau hang doi  int top(Queue \*Q){  return Q->data[Q->front];  }  void pop(Queue \*Q){  Q->front++;  }  List breath\_first\_search(Graph \*G, int x, int parent[]){  Queue Q;  make\_null\_Queue(&Q);  int mark[MAX\_Vertices];  int i;  for(i=1;i<=G->n;i++)  mark[i] = 0;  push\_Queue(&Q, x);  parent[x] = -1;  List L\_bfs;  make\_null(&L\_bfs);  while(!empty\_Queue(&Q)){  int u = top(&Q);  pop(&Q);  if(mark[u] == 1)  continue;  push\_back(&L\_bfs, u);  mark[u] = 1;  List L;  make\_null(&L);  L = neighbours(G, u);  int v;  for(i=1;i<=L.size;i++){  v = element\_at(&L, i);  if(mark[v] == 0){  push\_Queue(&Q, v);  if(parent[v] == -1)  parent[v] = u;  }  }  }  return L\_bfs;  }  int main(){  Graph G;  // freopen("BFS\_data.txt", "r", stdin);  int n,m,i,j;  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int x, y, e;  for(e=1;e<=m;e++){  scanf("%d%d",&x,&y);  add\_edge(&G, x , y);  }  int mark\_bfs[MAX\_Vertices], parent[MAX\_Vertices];  for(i=1;i<=G.n;i++){  mark\_bfs[i] = 0;  parent[i] = -1;  }  for(i=1;i<=G.n;i++){  if(mark\_bfs[i] == 0){  List L = breath\_first\_search(&G, i, parent);  for(j=1;j<=L.size;j++){  int v = element\_at(&L, j);  // printf("%d\n", v);  mark\_bfs[v] = 1;  }  }  printf("%d %d\n", i ,parent[i]);  }  } |
| Q2.  #include<stdio.h>  #define MAX\_Vertices 20  #define MAX\_Length 20  #define MAX\_Element 40  typedef struct{  int A[MAX\_Vertices][MAX\_Vertices];  int n; //So luong dinh  }Graph;  //Khoi tao do thi  void init\_Graph(Graph \*G, int n){  int i,j;  G->n = n;  for(i=1;i<=G->n;i++)//dong cua ma tran  for(j=1;j<=G->n;j++)//cot cua ma tran  G->A[i][j] = 0;  }  //Them cung vao do thi  void add\_edge(Graph \*G, int x, int y){  G->A[x][y] = 1;  G->A[y][x] = 1;  }  //Kiem tra dinh x va dinh y co phai la lang gieng cua nhau hay khong  int adjacent(Graph \*G, int x, int y){  return (G->A[x][y] != 0);  }  //Tinh bac cua dinh x trong do thi  int degree(Graph \*G, int x){  int deg = 0, i;  for(i=1;i<=G->n;i++)  if(adjacent(G, i, x))  deg++;  return deg;  }  //Khai bao cau truc danh sach List  typedef struct{  int data[MAX\_Length];  int size;  }List;  //Ham khoi tao List rong  void make\_null(List \*L){  L->size = 0;  }  //Them mot phan tu (dinh) vao danh sach  void push\_back(List \*L, int x){  L->data[L->size] = x;  L->size++;  }  //Lay mot phan tu (dinh) trong danh sach tai vi tri i  int element\_at(List \*L, int i){  return L->data[i-1];  }  //Tim lang gieng cua dinh x  List neighbours(Graph \*G, int x){  List L;  int i;  make\_null(&L);  for(i=1;i<=G->n;i++)  if(G->A[i][x] == 1)  push\_back(&L, i);  return L;  }  //Khai bao cau truc ngan xep  typedef struct{  int data[MAX\_Element];  int size;  }Stack;  //Khoi tao ngan xep rong  void make\_null\_stack(Stack \*S){  S->size = 0;  }  //Them mot phan tu vao trong ngan xep  void push\_stack(Stack \*S, int x){  S->data[S->size] = x;  S->size++;  }  //Lay mot phan tu trong Stack  int top(Stack \*S){  return S->data[S->size-1];  }  //Xoa mot phan tu thuoc Stack  void pop(Stack \*S){  S->size--;  }  //Kiem tra Stack co rong hay khong  int empty(Stack \*S){  return S->size == 0;  }  List depth\_first\_search(Graph \*G, int x, int parent[]){  Stack S;  make\_null\_stack(&S);  push\_stack(&S, x);  parent[x] = 0;  List list\_dfs;  make\_null(&list\_dfs);  //Khoi tao cac dinh chua duoc duyet  int mark[MAX\_Vertices]; //Danh dau xem danh do duoc duyet hay chua  int i;  for(i=1;i<=G->n;i++)  mark[i] = 0;  while(!empty(&S)){  int u = top(&S);  pop(&S);  if(mark[u] == 1)  continue;  // printf("Duyet: %d\n",u);  push\_back(&list\_dfs, u);  mark[u] = 1;  List L = neighbours(G, u);  for(i=1;i<=L.size;i++){  int v = element\_at(&L, i);  if(mark[v] == 0){  push\_stack(&S,v);  // if(parent[v] == -1)  parent[v] = u;  }  }  }  return list\_dfs;  }  int main(){  Graph G;  int n,m,i,j;  // freopen("DFS.txt", "r", stdin);  scanf("%d%d",&n,&m);  init\_Graph(&G,n);  int u,v,e;  for(e=1;e<=m;e++){  scanf("%d%d",&u,&v);  add\_edge(&G,u,v);  }    int parent[MAX\_Vertices];  for(i=1;i<=G.n;i++){  parent[i] = -1;  }    int mark\_dfs[MAX\_Vertices];  for(i=1;i<=G.n;i++){  mark\_dfs[i] = 0;  }  for(i=1;i<=G.n;i++){  if(mark\_dfs[i] == 0){  List dfs = depth\_first\_search(&G, i, parent);  for(j=1;j<=dfs.size;j++){  int k = element\_at(&dfs, j);  //printf("Duyet: %d\n", k);  mark\_dfs[k] = 1;  }  }  }  for(u=1;u<=G.n;u++)  printf("%d %d\n", u, parent[u]);  return 0;  } |
| Q3.  #include <stdio.h>  #define MAX\_N 100  typedef struct {  int n, m;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;  pG->m = 0;  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;  if (u != v)  pG->A[v][u] += 1;  pG->m++;  }  int adjacent(Graph \*pG, int u, int v) {  return pG->A[u][v] > 0;  }  int mark[MAX\_N];  int parent[MAX\_N];  void DFS(Graph \*pG, int u, int p) {  //1. Đánh dấu u đã duyệt  //printf("Duyet %d\n", u); //Làm gì đó trên u  mark[u] = 1; //Đánh dấu nó đã duyệt  parent[u] = p; //Cho cha của u là p  //2. Xét các đỉnh kề của u  for (int v = 1; v <= pG->n; v++)  if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt  DFS(pG, v, u); //gọi đệ quy duyệt nó  }  int main(){  Graph G;  int n, m, u, v;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (int e = 0; e < m; e++){  scanf("%d%d", &u, &v);  add\_edge(&G, u, v);  }  for (int u = 1; u <= G.n; u++) {  mark[u] = 0;  parent[u] = 0;  }  for (int u = 1; u <= G.n; u++)  if (mark[u] == 0)  DFS(&G, u, 0);  for (int u = 1; u <= G.n; u++)  printf("%d %d\n", u, parent[u]);  return 0;  } |

1. **BS - Đồ thị liên thông - Qua đảo**

|  |
| --- |
| #include <stdio.h>  /\* Khai báo CTDL Graph\*/  #define MAX\_N 100  typedef struct {  int n, m;  int A[MAX\_N][MAX\_N];  } Graph;  void init\_graph(Graph \*pG, int n) {  pG->n = n;  pG->m = 0;  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;  if (u != v)  pG->A[v][u] += 1;    pG->m++;  }  int adjacent(Graph \*pG, int u, int v) {  return pG->A[u][v] > 0;  }  //Biến hỗ trợ dùng để lưu trạng thái của đỉnh: đã duyệt/chưa duyệt  int mark[MAX\_N];  void DFS(Graph \*pG, int u) {  //1. Đánh dấu u đã duyệt  //printf("Duyet %d\n", u); //Làm gì đó trên u  mark[u] = 1; //Đánh dấu nó đã duyệt  //2. Xét các đỉnh kề của u  for (int v = 1; v <= pG->n; v++)  if (adjacent(pG, u, v) && mark[v] == 0) //Nếu v chưa duyệt  DFS(pG, v); //gọi đệ quy duyệt nó  }  int connected(Graph \*pG) {  int u;  for (u = 1; u <= pG->n; u++)  mark[u] = 0;  DFS(pG, 1);  for (u = 1; u <= pG->n; u++)  if (mark[u] == 0)  return 0;  return 1;  }  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++) {  mark[u] = 0;  }  if (connected(&G))  printf("YES\n");  else  printf("NO\n");  return 0;  } |

1. **BS - Tôn Ngộ Không.**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 100  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->A[v][u] += 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int mark[MAX\_N];  void DFS(Graph \*G, int u){  if(mark[u] == 1) return;  mark[u] = 1;  for(int v = 1;v<=G->n;v++)  if(adjacent(G,u,v) && !mark[v])  DFS(G,v);  }  int connected(Graph \*G){  int u;  for(u=1;u<=G->n;u++)  mark[u] = 0;  DFS(G,1);  for(u=1;u<=G->n;u++)  if(mark[u] == 0)  return 0;  return 1;  }  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++)  mark[u] = 0;  if(connected(&G))  printf("DUOC\n");  else printf("KHONG\n");  } |