**CODE FULL LÝ THUYẾT ĐỒ THỊ CT175**

& - &

1. ***Thực hành buổi 1:***
2. **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");  }  } |

1. ***Thực hành buổi 2:***
2. **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[u] = -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");  } |

1. ***Thực hành buổi 3:***
2. **BT1 - Thuật toán Moore - Dijkstra (pi và p)**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  MooreDijkstra(&G, 1);  for(u = 1; u <= G.n; u++)  printf("pi[%d] = %d, p[%d] = %d\n", u, pi[u], u, p[u]);  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->A[v][u] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  MooreDijkstra(&G, 1);  for(u = 1; u <= G.n; u++)  printf("pi[%d] = %d, p[%d] = %d\n", u, pi[u], u, p[u]);  } |

1. **BT2 - Thuật toán Moore - Dijkstra (chiều dài)**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  MooreDijkstra(&G, 1);  if(pi[n] < oo)  printf("%d",pi[n]);  else printf("-1");  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->A[v][u] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  MooreDijkstra(&G, 1);  if(pi[n] < oo)  printf("%d",pi[n]);  else printf("-1");  } |

1. **BT3 - Thuật toán Moore - Dijkstra (đường đi)**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  int s, t;  scanf("%d%d",&s,&t);  MooreDijkstra(&G, s);  int path[MAX\_N], k = 0, current = t;  while(current != -1){  path[k] = current; k++;  current = p[current];  }  for(u = k - 1; u >= 0; u--){  printf("%d ",path[u]);  if(u != 0)  printf("-> ");  }  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->A[v][u] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  int s, t;  scanf("%d%d",&s,&t);  MooreDijkstra(&G, s);  int path[MAX\_N], k = 0, current = t;  while(current != -1){  path[k] = current; k++;  current = p[current];  }  for(u = k - 1; u >= 0; u--){  printf("%d ",path[u]);  if(u != 0)  printf("-> ");  }  } |

1. **BT4 - Mê cung số**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N], mecung[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v;  scanf("%d%d",&m,&n);  init\_graph(&G, n\*m);  for(int i = 0; i < m; i++)  for(int j = 0; j < n; j++){  scanf("%d ",&u);  mecung[i\*n+j+1] = u;  }  for(int i = 0; i < m; i++)  for(int j = 0; j < n; j++){  int di[] = {-1,1,0,0};  int dj[] = {0,0,-1,1};  for(int k = 0; k < 4; k++){  int i\_ke = i + di[k];  int j\_ke = j + dj[k];  if( (i\_ke >= 0) && (i\_ke < m) && (j\_ke >= 0) && (j\_ke < n) ){  v = i\_ke \* n + j\_ke + 1;  u = i \* n + j + 1;  G.A[u][v] = mecung[v];  }  }  }  MooreDijkstra(&G, 1);  printf("%d",pi[G.n]);  } |

1. **BT5 - Ô kiều (Ngưu Lang - Chức Nữ)** (nằm ở BT bổ sung 2)
2. **BT6 - Thuật toán Bellman - Ford**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  typedef struct{  int u, v, w;  }Edge;  typedef struct{  Edge edge[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, int w){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->m++;  }  int pi[MAX\_N], p[MAX\_N];  void BellmanFord(Graph \*G, int s){  int u, v, w;  for(u = 1; u <= G->n; u++)  pi[u] = oo;  pi[s] = 0;  p[s] = -1;  for(int it = 1; it < G->n; it++)  for(int k = 0; k < G->m; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] == oo)  continue;  if(pi[u] + w < pi[v]){  pi[v] = pi[u] + w;  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  int s, t;  scanf("%d%d",&s,&t);  BellmanFord(&G, s);  if(pi[t] < oo)  printf("%d", pi[t]);  else printf("-1");  } |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  typedef struct{  int u, v, w;  }Edge;  typedef struct{  Edge edge[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, int w){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->m++;  }  int pi[MAX\_N], p[MAX\_N];  void BellmanFord(Graph \*G, int s){  int u, v, w;  for(u = 1; u <= G->n; u++)  pi[u] = oo;  pi[s] = 0;  p[s] = -1;  for(int it = 1; it < G->n; it++)  for(int k = 0; k < G->m; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] == oo)  continue;  if(pi[u] + w < pi[v]){  pi[v] = pi[u] + w;  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  int s, t;  scanf("%d%d",&s,&t);  BellmanFord(&G, s);  int path[MAX\_N], k = 0, current = t;  while(current != -1){  path[k] = current; k++;  current = p[current];  }  for(u = k - 1; u >= 0; u--){  printf("%d ", path[u]);  if(u != 0)  printf("-> ");  }  } |

1. **BT7 - Thuật toán Bellman - Ford (kiểm tra chu trình âm)**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 50  #define oo 9999  typedef struct{  int u, v, w;  }Edge;  typedef struct{  Edge edge[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, int w){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->m++;  }  int pi[MAX\_N], p[MAX\_N];  int BellmanFord(Graph \*G, int s){  int u, v, w;  for(u = 1; u <= G->n; u++)  pi[u] = oo;  pi[s] = 0;  p[s] = -1;  for(int it = 1; it < G->n; it++)  for(int k = 0; k < G->m; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] == oo)  continue;  if(pi[u] + w < pi[v]){  pi[v] = pi[u] + w;  p[v] = u;  }  }  for(int k = 0; k < G->m; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] == oo)  continue;  if(pi[u] + w < pi[v]){  return 1;  }  }  return 0;  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  int s;  scanf("%d",&s);  if(BellmanFord(&G, s))  printf("YES");  else printf("NO");    } |

1. **BT8 - Extended traffic** (Bài này bị đóng)
2. **BT9 - Thuật toán Floyd - Warshall (đường đi ngắn nhất giữa các cặp đỉnh)**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 40  #define oo 9999  #define NO\_EDGE 0  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;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int pi[MAX\_N][MAX\_N], next[MAX\_N][MAX\_N];  void FloydWarshall(Graph \*G){  int u, v, k;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  for(u = 1; u <= G->n; u++)  pi[u][u] = 0;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  if(G->A[u][v] != NO\_EDGE){  pi[u][v] = G->A[u][v];  next[u][v] = v;  }  }  for (k = 1; k <= G->n; k++)  for (u = 1; u <= G->n; u++) {  if (pi[u][k] == oo)  continue;  for (v = 1; v <= G->n; v++){  if (pi[k][v] == oo)  continue;  if (pi[u][k] + pi[k][v] < pi[u][v]){  pi[u][v] = pi[u][k] + pi[k][v];  next[u][v] = next[u][k];  }  }  }  }  int main(){  Graph G;  int u, v, n, m, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++)  for(v = 1; v <= G.n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  FloydWarshall(&G);  for(u = 1; u <= G.n; u++)  for(v = 1; v <= G.n; v++){  printf("%d -> %d: ", u, v);  if(pi[u][v] == oo || pi[u][v] > 9990)  printf("NO PATH\n");  else printf("%d\n", pi[u][v]);  }  return 0;  } |
| Q2.  #include<stdio.h>  #define MAX\_N 40  #define oo 9999  #define NO\_EDGE 0  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;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int pi[MAX\_N][MAX\_N], next[MAX\_N][MAX\_N];  void FloydWarshall(Graph \*G){  int u, v, k;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  for(u = 1; u <= G->n; u++)  pi[u][u] = 0;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  if(G->A[u][v] != NO\_EDGE){  pi[u][v] = G->A[u][v];  next[u][v] = v;  }  }  for (k = 1; k <= G->n; k++)  for (u = 1; u <= G->n; u++) {  if (pi[u][k] == oo)  continue;  for (v = 1; v <= G->n; v++){  if (pi[k][v] == oo)  continue;  if (pi[u][k] + pi[k][v] < pi[u][v]){  pi[u][v] = pi[u][k] + pi[k][v];  next[u][v] = next[u][k];  }  }  }  }  int main(){  Graph G;  int u, v, n, m, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++)  for(v = 1; v <= G.n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  FloydWarshall(&G);  for(u = 1; u <= G.n; u++){  for(v = 1; v <= G.n; v++){  if(pi[u][v] < oo){  printf("path(%d, %d): %d", u, v, u);  int current = u;  while(current != v){  current = next[current][v];  printf(" -> %d", current);  }  printf("\n");  }  else printf("path(%d, %d): NO PATH\n", u, v);  }  }  return 0;  } |

1. **BT10 - Thuật toán Floyd - Warshall (kiểm tra chu trình âm)**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  #define oo 9999  #define NO\_EDGE 0  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;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int pi[MAX\_N][MAX\_N], next[MAX\_N][MAX\_N];  int FloydWarshall(Graph \*G){  int u, v, k;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  for(u = 1; u <= G->n; u++)  pi[u][u] = 0;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  if(G->A[u][v] != NO\_EDGE){  pi[u][v] = G->A[u][v];  next[u][v] = v;  }  }  for (k = 1; k <= G->n; k++)  for (u = 1; u <= G->n; u++) {  if (pi[u][k] == oo)  continue;  for (v = 1; v <= G->n; v++){  if (pi[k][v] == oo)  continue;  if (pi[u][k] + pi[k][v] < pi[u][v]){  pi[u][v] = pi[u][k] + pi[k][v];  next[u][v] = next[u][k];  }  }  }  for(u = 1; u <= G->n; u++)  if(pi[u][u] < 0){  return 1;  }  return 0;  }  int main(){  Graph G;  int u, v, n, m, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++)  for(v = 1; v <= G.n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  if(FloydWarshall(&G) == 1)  printf("YES");  else printf("NO");    return 0;  } |

\*BT bổ sung:

1. **BT 1 - 2 (Tìm đường đi ngắn nhất)**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  MooreDijkstra(&G, 1);  if(pi[n] < oo)  printf("%d",pi[n]);  else printf("-1");  } |

|  |
| --- |
| Q2.  #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  mark[u] = 0;  }  MooreDijkstra(&G, 1);  if(pi[n] < oo)  printf("%d",pi[n]);  else printf("-1");  } |

1. **Kiểm tra chu trình âm và ứng dụng đường đi ngắn nhất**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX\_N 1000  #define oo 9999  typedef struct{  int u, v, w;  }Edge;  typedef struct{  int n, m;  Edge edge[MAX\_N];  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->m++;  }  int pi[MAX\_N], p[MAX\_N], negative\_cycle = 0;;  void BellmanFord(Graph \*G, int s){  int u, v, w, it, k;  for(u = 1; u <= G->n; u++)  pi[u] = oo;  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  for(k = 1; k <= G->n; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] == oo)  continue;  if(pi[u] + w < pi[v]){  pi[v] = pi[u] + w;  p[v] = u;  }  }  }  for(k = 1; k <= G->n; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] + w < pi[v]){  negative\_cycle = 1;  break;  }  }  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++){  pi[u] = oo;  }  negative\_cycle = 0;  BellmanFord(&G, 1);  if(negative\_cycle != 0)  printf("negative cycle");  else printf("ok");  } |
| Q2.  #include<stdio.h>  #define MAX\_N 40  #define oo 9999999  #define NO\_EDGE -1  typedef struct {  int A[MAX\_N][MAX\_N];  int m, n;  } 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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int pi[MAX\_N];  int p[MAX\_N];  int mark[MAX\_N];  void MooreDijkstra(Graph \*G, int s) {  for (int u = 1; u <= G->n; u++){  mark[u] = 0;  pi[u] = oo;  }  pi[s] = 0;  p[s] = -1;  for(int it = 1; it < G->n; it++){  int u, k, min\_pi = oo;  for(k = 1; k <= G->n; k++)  if(pi[k] < min\_pi && mark[k] == 0){  min\_pi = pi[k];  u = k;  }  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, w, e;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (e = 0; e < m; e++) {  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  for(u=1;u<=G.n;u++){  mark[u] = 0;  pi[u] = oo;  }  MooreDijkstra(&G, 1);  printf("%d",pi[n]);  } |

1. **Ứng dụng đường đi ngắn nhất**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  #define oo 9999999  #define NO\_EDGE -1  typedef struct {  int A[MAX\_N][MAX\_N];  int m, n;  } 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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->A[v][u] = w;  G->m++;  }  int pi[MAX\_N];  int p[MAX\_N];  int mark[MAX\_N];  void MooreDijkstra(Graph \*G, int s) {  for (int u = 1; u <= G->n; u++){  mark[u] = 0;  pi[u] = oo;  }  pi[s] = 0;  p[s] = -1;  for(int it = 1; it < G->n; it++){  int u, k, min\_pi = oo;  for(k = 1; k <= G->n; k++)  if(pi[k] < min\_pi && mark[k] == 0){  min\_pi = pi[k];  u = k;  }  mark[u] = 1;  int v;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v, w, e;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (e = 0; e < m; e++) {  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  for(u=1;u<=G.n;u++){  mark[u] = 0;  pi[u] = oo;  }  int s, t;  scanf("%d%d",&s,&t);  MooreDijkstra(&G, s);  printf("%d",pi[t]);  } |

1. **Mê cung số (Number Maze)**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 50  #define oo 9999  #define NO\_EDGE -1  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] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int mark[MAX\_N], pi[MAX\_N], p[MAX\_N], mecung[MAX\_N];  void MooreDijkstra(Graph \*G, int s){  int u, v, it;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  }  pi[s] = 0;  p[s] = -1;  for(it = 1; it < G->n; it++){  int j, min\_pi = oo;  for(j = 1; j <= G->n; j++)  if(mark[j] == 0 && pi[j] < min\_pi){  min\_pi = pi[j];  u = j;  }  mark[u] = 1;  for(v = 1; v <= G->n; v++)  if(G->A[u][v] != NO\_EDGE && mark[v] == 0)  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  p[v] = u;  }  }  }  int main(){  Graph G;  int n, m, u, v;  scanf("%d%d",&m,&n);  init\_graph(&G, n\*m);  for(int i = 0; i < m; i++)  for(int j = 0; j < n; j++){  scanf("%d ",&u);  mecung[i\*n+j+1] = u;  }  for(int i = 0; i < m; i++)  for(int j = 0; j < n; j++){  int di[] = {-1,1,0,0};  int dj[] = {0,0,-1,1};  for(int k = 0; k < 4; k++){  int i\_ke = i + di[k];  int j\_ke = j + dj[k];  if( (i\_ke >= 0) && (i\_ke < m) && (j\_ke >= 0) && (j\_ke < n) ){  v = i\_ke \* n + j\_ke + 1;  u = i \* n + j + 1;  G.A[u][v] = mecung[v];  }  }  }  MooreDijkstra(&G, 1);  printf("%d",pi[G.n]);  } |

1. **Bellman - Ford pi và p**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  #define oo 9999999  typedef struct{  int u, v, w;  }Edge;  typedef struct {  Edge edge[MAX\_N];  int m, n;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->m++;  }  int pi[MAX\_N];  int p[MAX\_N];  void BellmanFord(Graph \*G, int s) {  int u, v, w, k;  for (int u = 1; u <= G->n; u++){  pi[u] = oo;  }  pi[s] = 0;  p[s] = -1;  for(int it = 1; it < G->n; it++){  for(k = 0; k < G->m; k++){  u = G->edge[k].u;  v = G->edge[k].v;  w = G->edge[k].w;  if(pi[u] == oo)  continue;  if(pi[u] + w < pi[v]){  pi[v] = pi[u] + w;  p[v] = u;  }  }  }  }  int main(){  Graph G;  int n, m, u, v, w, e;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (e = 0; e < m; e++) {  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  for(u=1;u<=G.n;u++){  pi[u] = oo;  }  BellmanFord(&G, 1);  for(u=1;u<=G.n;u++)  printf("pi[%d] = %d, p[%d] = %d\n", u, pi[u], u, p[u]);  } |

1. **Floyd - Warshall**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  #define oo 9999  #define NO\_EDGE 0  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;  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->A[u][v] = NO\_EDGE;  }  void add\_edge(Graph \*G, int u, int v, int w){  G->A[u][v] = w;  G->m++;  }  int pi[MAX\_N][MAX\_N], next[MAX\_N][MAX\_N];  void FloydWarshall(Graph \*G){  int u, v, k;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  for(u = 1; u <= G->n; u++)  pi[u][u] = 0;  for(u = 1; u <= G->n; u++)  for(v = 1; v <= G->n; v++){  if(G->A[u][v] != NO\_EDGE){  pi[u][v] = G->A[u][v];  next[u][v] = v;  }  }  for (k = 1; k <= G->n; k++)  for (u = 1; u <= G->n; u++) {  if (pi[u][k] == oo)  continue;  for (v = 1; v <= G->n; v++){  if (pi[k][v] == oo)  continue;  if (pi[u][k] + pi[k][v] < pi[u][v]){  pi[u][v] = pi[u][k] + pi[k][v];  next[u][v] = next[u][k];  }  }  }  }  int main(){  Graph G;  int u, v, n, m, e, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w);  }  for(u = 1; u <= G.n; u++)  for(v = 1; v <= G.n; v++){  pi[u][v] = oo;  next[u][v] = -1;  }  FloydWarshall(&G);  for(u = 1; u <= G.n; u++)  for(v = 1; v <= G.n; v++){  printf("%d -> %d: ", u, v);  if(pi[u][v] == oo || pi[u][v] > 9990)  printf("NO PATH\n");  else printf("%d\n", pi[u][v]);  }  return 0;  } |

1. **Bellman - Ford**

|  |
| --- |
| #include <stdio.h>  #define MAX\_N 100  #define oo 999999  #define NO\_EDGE -1  typedef struct {  int u, v;  int w;  } Edge;  typedef struct {  int n, m;  Edge edges[MAX\_N];  } Graph;  void init\_graph(Graph \*G, int n) {  G->n = n;  G->m = 0;  }  void add\_edge(Graph \*G, int u, int v, int w) {  G->edges[G->m].u = u;  G->edges[G->m].v = v;  G->edges[G->m].w = w;  G->m++;  }  int pi[MAX\_N];  int p[MAX\_N];  int BellmanFord(Graph \*G, int s) {  for (int u = 1; u <= G->n; u++) {  pi[u] = oo;  }  pi[s] = 0;  p[s] = -1;  for (int it = 1; it < G->n; it++) {  for (int k = 0; k < G->m; k++) {  int u = G->edges[k].u;  int v = G->edges[k].v;  int w = G->edges[k].w;  if (pi[u] == oo)  continue;  if (pi[u] + w < pi[v]) {  pi[v] = pi[u] + w;  p[v] = u;  }  }  }  for (int k = 0; k < G->m; k++) {  int u = G->edges[k].u;  int v = G->edges[k].v;  int w = G->edges[k].w;  if (pi[u] == oo)  continue;  if (pi[u] + w < pi[v]) {  return 1;  }  }  return 0;  }  int main() {  Graph G;  int n, m, e, u, v, w;  scanf("%d%d", &n, &m);  init\_graph(&G, n);  for (e = 0; e < m; e++) {  scanf("%d%d%d", &u, &v, &w);  add\_edge(&G, u, v, w);  }  int s;  scanf("%d", &s);  if (BellmanFord(&G, s) == 1)  printf("YES\n");  else  printf("NO\n");  return 0;  } |

1. **Tìm số đường đi ngắn nhất (nâng cao)**

|  |
| --- |
| #include<stdio.h>  #define MAX 50  #define oo 9999  #define NO\_EDGE -1  typedef struct{  int A[MAX][MAX], n, m;  }Graph;  void init(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] = NO\_EDGE;  }  void add(Graph \*G, int u, int v, int w){  if( (u < 1 || u > G->n) || (v < 1 || v > G->n) || w < 0) return;  G->A[u][v] = w;  G->A[v][u] = w;  G->m++;  }  int pi[MAX];  int cnt[MAX];  int mark[MAX];  void dijkstra(Graph \*G, int s, int t){  for(int u = 1; u <= G->n; u++){  pi[u] = oo;  mark[u] = 0;  cnt[u] = 0;  }  pi[s] = 0;  for(int k = 1; k < G->n; k++){  int min\_pi = oo, u;  for(int v = 1; v <= G->n; v++){  if(!mark[v] && pi[v] < min\_pi){  min\_pi = pi[v];  u = v;  }  }  mark[u] = 1;  cnt[u] = 1;  for(int v = 1; v <= G->n; v++){  if(!mark[v] && G->A[u][v] != NO\_EDGE){  if(pi[u] + G->A[u][v] < pi[v]){  pi[v] = pi[u] + G->A[u][v];  cnt[v] = cnt[u];  }  else if(pi[u] + G->A[u][v] == pi[v]){  cnt[v] += cnt[u];  }  }  }  }  if(pi[t] == oo)  puts("-1 0");  else printf("%d %d\n", pi[t], cnt[t]);  }  int main(){  Graph G;  int n, m, u, v, e, w;  scanf("%d%d", &n, &m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add(&G, u, v, w);  }  dijkstra(&G, 1, n);  return 0;  } |

1. ***Thực hành buổi 4:***
2. **001. Thứ tự topo (chiều rộng)**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 40  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++;  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null\_List(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->danhsach[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  typedef struct{  int hangdoi[MAX\_N];  int front, rear;  }Queue;  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int x){  Q->rear++;  Q->hangdoi[Q->rear] = x;  }  void deQueue(Queue \*Q){  Q->front++;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  int emptyQueue(Queue \*Q){  return Q->front > Q->rear;  }  int d[MAX\_N];  void topo\_sort(Graph \*G, List \*L){  int d[100];  for(int u = 1; u <= G->n; u++){  d[u] = 0;  for(int x = 1; x <= G->n; x++)  if(G->A[x][u] != 0)  d[u]++;  }  Queue Q;  make\_null\_Queue(&Q);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  enQueue(&Q, u);  make\_null\_List(L);  while(!emptyQueue(&Q)){  int u = front(&Q); deQueue(&Q);  push\_back(L, u);  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] > 0){  d[v]--;  if(d[v] == 0)  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);  }  List L;  make\_null\_List(&L);  topo\_sort(&G, &L);  for(u = 1; u <= L.size; u++)  printf("%d ", element\_at(&L, u));  } |

1. **001b. Thứ tự topo (chiều rộng hoặc chiều sâu)**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 50  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }Graph;  void init(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++;  }  typedef struct{  int danhsach[MAX\_N], size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  int mark[MAX\_N];  void topo\_sort(Graph \*G, int u, List \*L){  if(mark[u] == 1) return;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] > 0 && mark[v] == 0)  topo\_sort(G, v, L);  push\_back(L, u);  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  List L;  make\_null(&L);  for(u = 1; u <= G.n; u++)  mark[u] = 0;  for(u = 1; u <= G.n; u++)  if (mark[u] == 0)  topo\_sort(&G, u, &L);  for(u = G.n; u >= 1; u--)  printf("%d ", element\_at(&L, u));  } |

1. **002. Xếp đá**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 50  typedef struct{  int A[MAX\_N][MAX\_N], n, m;  }Graph;  void init(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++;  }  typedef struct{  int danhsach[MAX\_N], size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  int mark[MAX\_N];  void topo\_sort(Graph \*G, int u, List \*L){  if(mark[u] == 1) return;  mark[u] = 1;  for(int v = 1; v <= G->n; v++)  if(G->A[u][v] > 0 && mark[v] == 0)  topo\_sort(G, v, L);  push\_back(L, u);  }  int main(){  Graph G;  int n, m, u, v, e;  scanf("%d%d",&n,&m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  List L;  make\_null(&L);  for(u = 1; u <= G.n; u++)  mark[u] = 0;  for(u = 1; u <= G.n; u++)  if (mark[u] == 0)  topo\_sort(&G, u, &L);  for(u = G.n; u >= 1; u--)  printf("%d\n", element\_at(&L, u));  } |

1. **003. Xếp hạng các đỉnh**

|  |
| --- |
| 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;  }  int indegree(Graph \*G, int u){  int v, deg = 0;  for(v = 1; v <= G->n; v++){  deg += G->A[v][u];  }  return deg + G->A[v][v];  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  void copy\_list(List \*L1, List \*L2){  int i, x;  make\_null(L1);  for(i = 1; i <= L2->size; i++){  x = element\_at(L2, i);  push\_back(L1, x);  }  }  List neighbours(Graph \*G, int u){\  List L;  make\_null(&L);  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G,u,v))  push\_back(&L, v);  }  return L;  }  int d[MAX\_N];  int rank[MAX\_N];  void RankGraph(Graph \*G){  int u;  for(u = 1; u <= G->n; u++){  d[u] = indegree(G, u);  }  List L1, L2;  make\_null(&L1);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&L1, u);  int k = 0;  while(L1.size > 0){  make\_null(&L2);  for(int i = 1; i <= L1.size; i++){  int u = element\_at(&L1, i);  rank[u] = k;  // push\_back(&L3, rank[u]);  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  d[v]--;  if(d[v] == 0)  push\_back(&L2, v);  }  }  }  copy\_list(&L1, &L2);  k++;  }  }  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);  }  RankGraph(&G);  for(u = 1; u <= G.n; u++)  printf("r[%d] = %d\n", u, rank[u]);  } |
| 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;  }  int indegree(Graph \*G, int u){  int v, deg = 0;  for(v = 1; v <= G->n; v++){  deg += G->A[v][u];  }  return deg + G->A[v][v];  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  void copy\_list(List \*L1, List \*L2){  int i;  make\_null(L1);  for(i = 1; i <= L2->size; i++){  push\_back(L1, element\_at(L2, i));  }  }  List neighbours(Graph \*G, int u){\  List L;  make\_null(&L);  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G,u,v))  push\_back(&L, v);  }  return L;  }  int d[MAX\_N];  int rank[MAX\_N];  void RankGraph(Graph \*G){  int u;  for(u = 1; u <= G->n; u++){  d[u] = indegree(G, u);  }  List L1, L2;  make\_null(&L1);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&L1, u);  int k = 0;  while(L1.size > 0){  make\_null(&L2);  for(int i = 1; i <= L1.size; i++){  int u = element\_at(&L1, i);  rank[u] = k;  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  d[v]--;  if(d[v] == 0)  push\_back(&L2, v);  }  }  }  copy\_list(&L1, &L2);  k++;  }  }  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);  }  RankGraph(&G);  for(u = 1; u <= G.n; u++)  printf("r[%d] = %d\n", u, rank[u]);  } |

1. **004. Ứng dụng xếp hạng**

|  |
| --- |
| 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[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int indegree(Graph \*G, int u){  int v, deg = 0;  for(v = 1; v <= G->n; v++){  deg += G->A[v][u];  }  return deg + G->A[v][v];  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  void copy\_list(List \*L1, List \*L2){  int i, x;  make\_null(L1);  for(i = 1; i <= L2->size; i++){  x = element\_at(L2, i);  push\_back(L1, x);  }  }  List neighbours(Graph \*G, int u){  List L;  make\_null(&L);  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G,u,v))  push\_back(&L, v);  }  return L;  }  int d[MAX\_N];  int rank[MAX\_N];  void RankGraph(Graph \*G){  int u;  for(u = 1; u <= G->n; u++){  d[u] = 0;  for(int x = 1; x <= G->n; x++)  if(G->A[x][u] != 0)  d[u]++;  }  List L1, L2;  make\_null(&L1);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&L1, u);  int k = 0;  while(L1.size > 0){  make\_null(&L2);  for(int i = 1; i <= L1.size; i++){  int u = element\_at(&L1, i);  rank[u] = k;  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  d[v]--;  if(d[v] == 0)  push\_back(&L2, v);  }  }  }  copy\_list(&L1, &L2);  k++;  }  }  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);  }  RankGraph(&G);  int max = 0;  for(u = 1; u <= G.n; u++){  printf("%d\n", rank[u] + 1);  max += rank[u] + 1;  }  printf("%d",max);    } |
| 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;  }  int indegree(Graph \*G, int u){  int v, deg = 0;  for(v = 1; v <= G->n; v++){  deg += G->A[v][u];  }  return deg + G->A[v][v];  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  void copy\_list(List \*L1, List \*L2){  int i, x;  make\_null(L1);  for(i = 1; i <= L2->size; i++){  x = element\_at(L2, i);  push\_back(L1, x);  }  }  List neighbours(Graph \*G, int u){  List L;  make\_null(&L);  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G,u,v))  push\_back(&L, v);  }  return L;  }  int d[MAX\_N];  int rank[MAX\_N];  void RankGraph(Graph \*G){  int u;  for(u = 1; u <= G->n; u++){  d[u] = 0;  for(int x = 1; x <= G->n; x++)  if(G->A[x][u] != 0)  d[u]++;  }  List L1, L2;  make\_null(&L1);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&L1, u);  int k = 0;  while(L1.size > 0){  make\_null(&L2);  for(int i = 1; i <= L1.size; i++){  int u = element\_at(&L1, i);  rank[u] = k;  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  d[v]--;  if(d[v] == 0)  push\_back(&L2, v);  }  }  }  copy\_list(&L1, &L2);  k++;  }  }  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);  }  RankGraph(&G);  for(u = 1; u <= G.n; u++)  printf("%d ", rank[u] + 1);    } |

1. **005. Quản lý dự án**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 100  #define oo 9999999  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 <= 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->m++;  }  int adjacent(Graph \*G, int x, int y){  return G->A[x][y] != 0 ;  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null\_List(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->danhsach[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  List neighbors(Graph \*G, int x){  int y;  List list;  make\_null\_List(&list);  for(y = 1; y <= G->n; y++)  if(adjacent(G, x, y))  push\_back(&list, y);  return list;  }  void copy\_list(List \*L1,List \*L2){  make\_null\_List(L1);  int i;  for(i = 1; i <= L2->size; i++){  int u = element\_at(L2, i);  push\_back(L1, u);  }  }  typedef struct{  int front,rear;  int hangdoi[MAX\_N];  }Queue;  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int x){  Q->rear++;  Q->hangdoi[Q->rear] = x;  }  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 min(int a,int b){  return a < b ? a : b;  }  int max(int a , int b){  return a > b ? a : b;  }  int d[MAX\_N];  void topo\_sort(Graph \*G, List \*L){  int d[100];  int x, u;  Queue Q;  make\_null\_Queue(&Q);  for (u = 1; u <= G->n; u++)  d[u] = 0;  for (x = 1; x <= G->n; x++)  for (u = 1; u <= G->n; u++)  if (G->A[x][u] != 0)  d[u]++;  for(u = 1; u <= G->n; u++)  if(d[u] == 0)  enQueue(&Q, u);  make\_null\_List(L);  while(!empty(&Q)){  u = front(&Q);  deQueue(&Q);  push\_back(L,u);  List list = neighbors(G, u);  for(x = 1; x <= list.size; x++){  int v = element\_at(&list, x);  d[v]--;  if(d[v] == 0)  enQueue(&Q, v);  }  }  }  int main(){  Graph G;  int n, u, x, v, j;  List L;;  scanf("%d", &n);  init\_graph(&G, n+2);  int alpha = n + 1, beta = n + 2;  d[alpha] = 0;  for (u = 1; u <= n; u++) {  scanf("%d",&d[u]);  do {  scanf("%d", &x);  if (x > 0)  add\_edge(&G, x, u);  }while (x > 0);  }  for (u = 1; u <= n; u++) {  int deg\_neg = 0;  for (x = 1; x <= n; x++)  if (G.A[x][u] > 0)  deg\_neg++;  if (deg\_neg == 0)  add\_edge(&G, alpha, u);  }  for (u = 1; u <= n; u++) {  int deg\_pos = 0;  for (v = 1; v <= n; v++)  if (G.A[u][v] > 0)  deg\_pos++;  if (deg\_pos == 0)  add\_edge(&G, u, beta);  }  topo\_sort(&G ,&L);  int t[100];  t[alpha] = 0;  for (j = 2; j <= L.size; j++) {  u = element\_at(&L, j);  t[u] = 0;  for (x = 1; x <= G.n; x++)  if (G.A[x][u] > 0)  t[u] = max(t[u], t[x] + d[x]);  }  int T[100];  T[beta] = t[beta];  for (j = L.size - 1; j >= 1; j --) {  int u = element\_at(&L, j);  T[u] = oo;  for (v = 1; v <= G.n; v++)  if (G.A[u][v] > 0)  T[u] = min(T[u], T[v] - d[u]);  }  for(u = 1; u <= n; u++)  printf("%d %d\n",t[u],T[u]);  } |

1. **006. Quản lý dự án phần mềm**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 100  #define oo 9999999  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 <= 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->m++;  }  int adjacent(Graph \*G, int x, int y){  return G->A[x][y] != 0 ;  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null\_List(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->danhsach[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  List neighbors(Graph \*G, int x){  int y;  List list;  make\_null\_List(&list);  for(y = 1; y <= G->n; y++)  if(adjacent(G, x, y))  push\_back(&list, y);  return list;  }  void copy\_list(List \*L1,List \*L2){  make\_null\_List(L1);  int i;  for(i = 1; i <= L2->size; i++){  int u = element\_at(L2, i);  push\_back(L1, u);  }  }  typedef struct{  int front,rear;  int hangdoi[MAX\_N];  }Queue;  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int x){  Q->rear++;  Q->hangdoi[Q->rear] = x;  }  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 min(int a,int b){  return a < b ? a : b;  }  int max(int a , int b){  return a > b ? a : b;  }  int d[MAX\_N];  void topo\_sort(Graph \*G, List \*L){  int d[100];  int x, u;  Queue Q;  make\_null\_Queue(&Q);  for (u = 1; u <= G->n; u++)  d[u] = 0;  for (x = 1; x <= G->n; x++)  for (u = 1; u <= G->n; u++)  if (G->A[x][u] != 0)  d[u]++;  for(u = 1; u <= G->n; u++)  if(d[u] == 0)  enQueue(&Q, u);  make\_null\_List(L);  while(!empty(&Q)){  u = front(&Q);  deQueue(&Q);  push\_back(L,u);  List list = neighbors(G, u);  for(x = 1; x <= list.size; x++){  int v = element\_at(&list, x);  d[v]--;  if(d[v] == 0)  enQueue(&Q, v);  }  }  }  int main(){  Graph G;  int n, u, x, v, j, m, e;  List L;  scanf("%d", &n);  init\_graph(&G, n+2);  int alpha = n + 1, beta = n + 2;  d[alpha] = 0;  for (u = 1; u <= n; u++) {  scanf("%d",&d[u]);  }  scanf("%d",&m);  for(e = 1; e <= m; e++){  scanf("%d%d",&u,&v);  add\_edge(&G, u, v);  }  for (u = 1; u <= n; u++) {  int deg\_neg = 0;  for (x = 1; x <= n; x++)  if (G.A[x][u] > 0)  deg\_neg++;  if (deg\_neg == 0)  add\_edge(&G, alpha, u);  }  for (u = 1; u <= n; u++) {  int deg\_pos = 0;  for (v = 1; v <= n; v++)  if (G.A[u][v] > 0)  deg\_pos++;  if (deg\_pos == 0)  add\_edge(&G, u, beta);  }  topo\_sort(&G ,&L);  int t[100];  t[alpha] = 0;  for (j = 2; j <= L.size; j++) {  u = element\_at(&L, j);  t[u] = 0;  for (x = 1; x <= G.n; x++)  if (G.A[x][u] > 0)  t[u] = max(t[u], t[x] + d[x]);  }  int T[100];  T[beta] = t[beta];  for (j = L.size - 1; j >= 1; j --) {  int u = element\_at(&L, j);  T[u] = oo;  for (v = 1; v <= G.n; v++)  if (G.A[u][v] > 0)  T[u] = min(T[u], T[v] - d[u]);  }  printf("%d", T[beta]);  } |

\*BT bổ sung:

1. **Xếp hạng đồ thị (check được)**

|  |
| --- |
| #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;  }  int indegree(Graph \*G, int u){  int v, deg = 0;  for(v = 1; v <= G->n; v++){  deg += G->A[v][u];  }  return deg + G->A[v][v];  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  void copy\_list(List \*L1, List \*L2){  int i, x;  make\_null(L1);  for(i = 1; i <= L2->size; i++){  x = element\_at(L2, i);  push\_back(L1, x);  }  }  List neighbours(Graph \*G, int u){\  List L;  make\_null(&L);  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G,u,v))  push\_back(&L, v);  }  return L;  }  int d[MAX\_N];  int rank[MAX\_N];  int S[MAX\_N];  void RankGraph(Graph \*G){  int u;  for(u = 1; u <= G->n; u++){  d[u] = indegree(G, u);  }  List L1, L2;  make\_null(&L1);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&L1, u);  int k = 0;  while(L1.size > 0){  make\_null(&L2);  for(int i = 1; i <= L1.size; i++){  int u = element\_at(&L1, i);  rank[u] = k;  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  d[v]--;  if(d[v] == 0)  push\_back(&L2, v);  }  }  }  copy\_list(&L1, &L2);  k++;  }  }  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);  }  RankGraph(&G);  for(u = 1; u <= G.n; u++)  printf("%d\n", rank[u]);  } |

1. **Cân đá**

|  |
| --- |
| #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 danhsach[MAX\_N];  int size;  }List;  void make\_null\_List(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->danhsach[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  int d[MAX\_N], mark[MAX\_N];  void topo\_sort(Graph \*G, int u, List \*L){  if(mark[u] == 1) return;  mark[u] = 1;  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  if(!mark[v])  topo\_sort(G, v, L);  }  }  push\_back(L, u);  }  int main(){  Graph G;  int u, v, n, m, 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;  List L;  for(u = 1; u <= G.n; u++)  if(mark[u] == 0)  topo\_sort(&G, u, &L);  for(u = L.size; u >= 1; u--)  printf("%d \n", element\_at(&L, u));  } |

1. **Chia kẹo**

|  |
| --- |
| #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[v][u] = 1;  G->m++;  }  int adjacent(Graph \*G, int u, int v){  return G->A[u][v] != 0;  }  int indegree(Graph \*G, int u){  int v, deg = 0;  for(v = 1; v <= G->n; v++){  deg += G->A[v][u];  }  return deg + G->A[v][v];  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null(List \*L){  L->size = 0;  }  void push\_back(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  void copy\_list(List \*L1, List \*L2){  int i, x;  make\_null(L1);  for(i = 1; i <= L2->size; i++){  x = element\_at(L2, i);  push\_back(L1, x);  }  }  List neighbours(Graph \*G, int u){  List L;  make\_null(&L);  int v;  for(v = 1; v <= G->n; v++){  if(adjacent(G,u,v))  push\_back(&L, v);  }  return L;  }  int d[MAX\_N];  int rank[MAX\_N];  void RankGraph(Graph \*G){  int u;  for(u = 1; u <= G->n; u++){  d[u] = 0;  for(int x = 1; x <= G->n; x++)  if(G->A[x][u] != 0)  d[u]++;  }  List L1, L2;  make\_null(&L1);  for(int u = 1; u <= G->n; u++)  if(d[u] == 0)  push\_back(&L1, u);  int k = 0;  while(L1.size > 0){  make\_null(&L2);  for(int i = 1; i <= L1.size; i++){  int u = element\_at(&L1, i);  rank[u] = k;  for(int v = 1; v <= G->n; v++){  if(adjacent(G, u, v)){  d[v]--;  if(d[v] == 0)  push\_back(&L2, v);  }  }  }  copy\_list(&L1, &L2);  k++;  }  }  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);  }  RankGraph(&G);  int max = 0;  for(u = 1; u <= G.n; u++){  printf("%d\n", rank[u] + 1);  max += rank[u] + 1;  }  printf("%d",max);  } |

1. **Tổ chức thi công - Dự án xây nhà**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 100  #define oo 9999999  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 <= 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->m++;  }  int adjacent(Graph \*G, int x, int y){  return G->A[x][y] != 0 ;  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null\_List(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->danhsach[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  List neighbors(Graph \*G, int x){  int y;  List list;  make\_null\_List(&list);  for(y = 1; y <= G->n; y++)  if(adjacent(G, x, y))  push\_back(&list, y);  return list;  }  void copy\_list(List \*L1,List \*L2){  make\_null\_List(L1);  int i;  for(i = 1; i <= L2->size; i++){  int u = element\_at(L2, i);  push\_back(L1, u);  }  }  typedef struct{  int front,rear;  int hangdoi[MAX\_N];  }Queue;  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int x){  Q->rear++;  Q->hangdoi[Q->rear] = x;  }  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 min(int a,int b){  return a < b ? a : b;  }  int max(int a , int b){  return a > b ? a : b;  }  int d[MAX\_N];  void topo\_sort(Graph \*G, List \*L){  int d[100];  int x, u;  Queue Q;  make\_null\_Queue(&Q);  for (u = 1; u <= G->n; u++)  d[u] = 0;  for (x = 1; x <= G->n; x++)  for (u = 1; u <= G->n; u++)  if (G->A[x][u] != 0)  d[u]++;  for(u = 1; u <= G->n; u++)  if(d[u] == 0)  enQueue(&Q, u);  make\_null\_List(L);  while(!empty(&Q)){  u = front(&Q);  deQueue(&Q);  push\_back(L,u);  List list = neighbors(G, u);  for(x = 1; x <= list.size; x++){  int v = element\_at(&list, x);  d[v]--;  if(d[v] == 0)  enQueue(&Q, v);  }  }  }  int main(){  Graph G;  int n, u, x, v, j;  List L;;  scanf("%d", &n);  init\_graph(&G, n+2);  int alpha = n + 1, beta = n + 2;  d[alpha] = 0;  for (u = 1; u <= n; u++) {  scanf("%d",&d[u]);  do {  scanf("%d", &x);  if (x > 0)  add\_edge(&G, x, u);  }while (x > 0);  }  for (u = 1; u <= n; u++) {  int deg\_neg = 0;  for (x = 1; x <= n; x++)  if (G.A[x][u] > 0)  deg\_neg++;  if (deg\_neg == 0)  add\_edge(&G, alpha, u);  }  for (u = 1; u <= n; u++) {  int deg\_pos = 0;  for (v = 1; v <= n; v++)  if (G.A[u][v] > 0)  deg\_pos++;  if (deg\_pos == 0)  add\_edge(&G, u, beta);  }  topo\_sort(&G ,&L);  int t[100];  t[alpha] = 0;  for (j = 2; j <= L.size; j++) {  u = element\_at(&L, j);  t[u] = 0;  for (x = 1; x <= G.n; x++)  if (G.A[x][u] > 0)  t[u] = max(t[u], t[x] + d[x]);  }  int T[100];  T[beta] = t[beta];  for (j = L.size - 1; j >= 1; j --) {  int u = element\_at(&L, j);  T[u] = oo;  for (v = 1; v <= G.n; v++)  if (G.A[u][v] > 0)  T[u] = min(T[u], T[v] - d[u]);  }  printf("%d\n",t[beta]);  for(u = 1; u <= G.n; u++)  printf("%d-%d\n", t[u], T[u]);  } |

1. **Tổ chức thi công - Dự án phần mềm**

|  |
| --- |
| #include<stdio.h>  #define MAX\_N 100  #define oo 9999999  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 <= 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->m++;  }  int adjacent(Graph \*G, int x, int y){  return G->A[x][y] != 0 ;  }  typedef struct{  int danhsach[MAX\_N];  int size;  }List;  void make\_null\_List(List \*L){  L->size = 0;  }  void push\_back(List \*L, int x){  L->danhsach[L->size] = x;  L->size++;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  List neighbors(Graph \*G, int x){  int y;  List list;  make\_null\_List(&list);  for(y = 1; y <= G->n; y++)  if(adjacent(G, x, y))  push\_back(&list, y);  return list;  }  void copy\_list(List \*L1,List \*L2){  make\_null\_List(L1);  int i;  for(i = 1; i <= L2->size; i++){  int u = element\_at(L2, i);  push\_back(L1, u);  }  }  typedef struct{  int front,rear;  int hangdoi[MAX\_N];  }Queue;  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int x){  Q->rear++;  Q->hangdoi[Q->rear] = x;  }  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 min(int a,int b){  return a < b ? a : b;  }  int max(int a , int b){  return a > b ? a : b;  }  int d[MAX\_N];  void topo\_sort(Graph \*G, List \*L){  int d[100];  int x, u;  Queue Q;  make\_null\_Queue(&Q);  for (u = 1; u <= G->n; u++)  d[u] = 0;  for (x = 1; x <= G->n; x++)  for (u = 1; u <= G->n; u++)  if (G->A[x][u] != 0)  d[u]++;  for(u = 1; u <= G->n; u++)  if(d[u] == 0)  enQueue(&Q, u);  make\_null\_List(L);  while(!empty(&Q)){  u = front(&Q);  deQueue(&Q);  push\_back(L,u);  List list = neighbors(G, u);  for(x = 1; x <= list.size; x++){  int v = element\_at(&list, x);  d[v]--;  if(d[v] == 0)  enQueue(&Q, v);  }  }  }  int main(){  Graph G;  int n, u, x, v, j;  List L;;  scanf("%d", &n);  init\_graph(&G, n+2);  int alpha = n + 1, beta = n + 2;  d[alpha] = 0;  for (u = 1; u <= n; u++) {  scanf("%d",&d[u]);  do {  scanf("%d", &x);  if (x > 0)  add\_edge(&G, x, u);  }while (x > 0);  }  for (u = 1; u <= n; u++) {  int deg\_neg = 0;  for (x = 1; x <= n; x++)  if (G.A[x][u] > 0)  deg\_neg++;  if (deg\_neg == 0)  add\_edge(&G, alpha, u);  }  for (u = 1; u <= n; u++) {  int deg\_pos = 0;  for (v = 1; v <= n; v++)  if (G.A[u][v] > 0)  deg\_pos++;  if (deg\_pos == 0)  add\_edge(&G, u, beta);  }  topo\_sort(&G ,&L);  int t[100];  t[alpha] = 0;  for (j = 2; j <= L.size; j++) {  u = element\_at(&L, j);  t[u] = 0;  for (x = 1; x <= G.n; x++)  if (G.A[x][u] > 0)  t[u] = max(t[u], t[x] + d[x]);  }  int T[100];  T[beta] = t[beta];  for (j = L.size - 1; j >= 1; j --) {  int u = element\_at(&L, j);  T[u] = oo;  for (v = 1; v <= G.n; v++)  if (G.A[u][v] > 0)  T[u] = min(T[u], T[v] - d[u]);  }  int s, tg;  scanf("%d%d",&s,&tg);  if(tg < T[s])  printf("YES");  else printf("NO");  } |

1. ***Thực hành buổi 5:***
2. **BT5.1. Tìm cây khung bằng giải thuật Kruskal**

|  |
| --- |
| #include<stdio.h>  #define MAX 9999  typedef struct{  int u, v, w;  }Edge;  typedef struct{  Edge edges[MAX];  int n, m;  }Graph;  void init(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add(Graph \*G, int u, int v, int w){  G->edges[G->m].u = (u < v ? u : v);  G->edges[G->m].v = (u > v ? u : v);  G->edges[G->m].w = w;  G->m++;  }  int parent[MAX];  int findroot(int u){  while(parent[u] != u)  u = parent[u];  return u;  }  int Kruskal(Graph \*G, Graph \*T){  int u, v, w;  for(int i = 0; i < G->m; i++)  for(int j = i + 1; j < G->m; j++)  if(G->edges[i].w > G->edges[j].w){  Edge temp = G->edges[i];  G->edges[i] = G->edges[j];  G->edges[j] = temp;  }  init(T, G->n);  for(u = 1; u <= G->n; u++)  parent[u] = u;  int sum\_w = 0;  for(int e = 0; e < G->m; e++){  u = G->edges[e].u;  v = G->edges[e].v;  w = G->edges[e].w;  int root\_u = findroot(u);  int root\_v = findroot(v);  if(root\_u != root\_v){  add(T, u, v, w);  parent[root\_v] = root\_u;  sum\_w += w;  }  }  return sum\_w;  }  int main(){  Graph G, T;  int n, m, u, v, w, e;  scanf("%d%d",&n,&m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add(&G, u, v, w);  }  int sum\_w = Kruskal(&G, &T);  printf("%d\n", sum\_w);  for(e = 0; e < T.m; e++){  printf("%d %d %d\n", T.edges[e].u, T.edges[e].v, T.edges[e].w);  }  } |

1. **BT5.2. Tìm cây khung có trọng lượng nhỏ nhất bằng giải thuật Prim**

|  |
| --- |
| #include<stdio.h>  #define MAX 50  #define oo 9999  #define NO\_EDGE 0  typedef struct{  int A[MAX][MAX], n, m;  }Graph;  void init(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] = NO\_EDGE;  }  }  }  void add(Graph \*G, int u, int v, int w){  G->A[u][v] += w;  G->A[v][u] += w;  G->m++;  }  int pi[MAX];  int p[MAX];  int mark[MAX];  int Prim(Graph \*G, Graph \*T, int s){  int u, v, i, x;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  p[u] = -1;  mark[u] = 0;  }  pi[s] = 0;  for(i = 1; i < G->n; i++){  int min\_dist = oo;  for(x = 1; x <= G->n; x++){  if(!mark[x] && pi[x] < min\_dist){  min\_dist = pi[x];  u = x;  }  }  mark[u] = 1;  for(v = 1; v <= G->n; v++){  if(!mark[v] && G->A[u][v] != NO\_EDGE && pi[v] > G->A[u][v]){  pi[v] = G->A[u][v];  p[v] = u;  }  }  }  init(T, G->n);  int sum\_w = 0;  for(u = 1; u <= G->n; u++){  if(p[u] != -1){  int w = G->A[p[u]][u];  add(T, p[u], u, w);  sum\_w += w;  }  }  return sum\_w;  }  int main(){  Graph G, T;  int n, m, u, v, w, e;  scanf("%d%d",&n, &m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d", &u, &v, &w);  add(&G, u, v, w);  }  init(&T, n);  int sum\_w = 0;  sum\_w = Prim(&G, &T, 1);  printf("%d\n", sum\_w);  for(u = 1; u <= T.n; u++){  for(v = 1; v <= T.n; v++){  if(T.A[u][v] != NO\_EDGE && u < v)  printf("%d %d %d\n", u, v, T.A[u][v]);  }  }  } |

1. **BT5.3. Ứng dụng cây khung**

|  |
| --- |
| Q1.  #include<stdio.h>  #define MAX 9999  typedef struct{  int u, v, w;  }Edge;  typedef struct{  Edge edge[MAX];  int n, m;  }Graph;  void init(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add(Graph \*G, int u, int v, int w){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->m++;  }  int parent[MAX];  int findroot(int u){  while(parent[u] != u)  u = parent[u];  return u;  }  int Kruskal(Graph \*G, Graph \*T){  int u, v, w;  for(int i = 0; i < G->m; i++)  for(int j = i + 1; j < G->m; j++)  if(G->edge[i].w > G->edge[j].w){  Edge temp = G->edge[i];  G->edge[i] = G->edge[j];  G->edge[j] = temp;  }  init(T, G->n);  for(u = 1; u <= G->n; u++)  parent[u] = u;  int sum\_w = 0;  for(int e = 0; e < G->m; e++){  u = G->edge[e].u;  v = G->edge[e].v;  w = G->edge[e].w;  int root\_u = findroot(u);  int root\_v = findroot(v);  if(root\_u != root\_v){  add(T, u, v, w);  parent[root\_v] = root\_u;  sum\_w += w;  }  }  return sum\_w;  }  int main(){  Graph G, T;  int n, m, u, v, w, e;  scanf("%d%d",&n,&m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add(&G, u, v, w);  }  int sum\_w\_chuaxoa = 0;  for(e = 0; e < m; e++){  sum\_w\_chuaxoa += G.edge[e].w;  }  int sum\_w = Kruskal(&G, &T);  printf("%d\n", sum\_w\_chuaxoa - sum\_w);    } |
| Q2.  #include<stdio.h>  #define MAX 50  #define oo 9999  #define NO\_EDGE 0  typedef struct{  int A[MAX][MAX], n, m;  }Graph;  void init(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] = NO\_EDGE;  }  }  }  void add(Graph \*G, int u, int v, int w){  G->A[u][v] += w;  G->A[v][u] += w;  G->m++;  }  int pi[MAX];  int p[MAX];  int mark[MAX];  void Prim(Graph \*G, Graph \*T, int s){  int u, v, i, x;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  p[u] = -1;  mark[u] = 0;  }  pi[s] = 0;  for(i = 1; i <= G->n; i++){  int min\_dist = oo;  for(x = 1; x <= G->n; x++){  if(!mark[x] && pi[x] < min\_dist){  min\_dist = pi[x];  u = x;  }  }  mark[u] = 1;  printf("%d\n", u);  for(v = 1; v <= G->n; v++){  if(!mark[v] && G->A[u][v] != NO\_EDGE && pi[v] > G->A[u][v]){  pi[v] = G->A[u][v];  p[v] = u;  }  }  }  init(T, G->n);  int sum\_w = 0;  for(u = 1; u <= G->n; u++){  if(p[u] != -1){  int w = G->A[p[u]][u];  add(T, p[u], u, w);  sum\_w += w;  }  }  }  int main(){  Graph G, T;  int n, m, u, v, w, e;  scanf("%d%d",&n, &m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d", &u, &v, &w);  add(&G, u, v, w);  }  init(&T, n);  Prim(&G, &T, 1);  } |
| Q3.  #include<stdio.h>  #define MAX 50  #define oo 9999  #define NO\_EDGE 0  typedef struct{  int A[MAX][MAX], n, m;  }Graph;  void init(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] = NO\_EDGE;  }  }  }  void add(Graph \*G, int u, int v, int w){  G->A[u][v] += w;  G->A[v][u] += w;  G->m++;  }  int pi[MAX];  int p[MAX];  int mark[MAX];  int Prim(Graph \*G, Graph \*T, int s){  int u, v, i, x;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  p[u] = -1;  mark[u] = 0;  }  pi[s] = 0;  for(i = 1; i < G->n; i++){  int min\_dist = oo;  for(x = 1; x <= G->n; x++){  if(!mark[x] && pi[x] < min\_dist){  min\_dist = pi[x];  u = x;  }  }  mark[u] = 1;  for(v = 1; v <= G->n; v++){  if(!mark[v] && G->A[u][v] != NO\_EDGE && pi[v] > G->A[u][v]){  pi[v] = G->A[u][v];  p[v] = u;  }  }  }  init(T, G->n);  int sum\_w = 0;  for(u = 1; u <= G->n; u++){  if(p[u] != -1){  int w = G->A[p[u]][u];  add(T, p[u], u, w);  sum\_w += w;  }  }  return sum\_w;  }  int main(){  Graph G, T;  int n, m, u, v, w, e, k;  scanf("%d%d%d",&n, &m, &k);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d", &u, &v, &w);  add(&G, u, v, w);  }  init(&T, n);  int sum\_w = 0;  sum\_w = Prim(&G, &T, 1);  if(sum\_w <= k)  printf("OK");  else printf("%d",sum\_w - k);  } |
| Q4.  #include<stdio.h>  #define MAX 50  #define oo 9999  #define NO\_EDGE 0  typedef struct{  int A[MAX][MAX], n, m;  }Graph;  void init(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] = NO\_EDGE;  }  }  }  void add(Graph \*G, int u, int v, int w){  G->A[u][v] += w;  G->A[v][u] += w;  G->m++;  }  int pi[MAX];  int p[MAX];  int mark[MAX];  int Prim(Graph \*G, Graph \*T, int s){  int u, v, i, x;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  p[u] = -1;  mark[u] = 0;  }  pi[s] = 0;  for(i = 1; i < G->n; i++){  int min\_dist = oo;  for(x = 1; x <= G->n; x++){  if(!mark[x] && pi[x] < min\_dist){  min\_dist = pi[x];  u = x;  }  }  mark[u] = 1;  for(v = 1; v <= G->n; v++){  if(!mark[v] && G->A[u][v] != NO\_EDGE && pi[v] > G->A[u][v]){  pi[v] = G->A[u][v];  p[v] = u;  }  }  }  init(T, G->n);  int sum\_w = 0;  for(u = 1; u <= G->n; u++){  if(p[u] != -1){  int w = G->A[p[u]][u];  add(T, p[u], u, w);  sum\_w += w;  }  }  return sum\_w;  }  int main(){  Graph G, T;  int n, m, u, v, w, e;  scanf("%d%d",&n, &m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d", &u, &v, &w);  add(&G, u, v, w);  }  init(&T, n);  int sum\_w = 0;  sum\_w = Prim(&G, &T, 1);  printf("%d\n", sum\_w);  } |
| Q5.  #include<stdio.h>  #define MAX 50  #define oo 9999  typedef struct{  int u, v, w, d;  }Edge;  typedef struct{  Edge edge[MAX];  int n, m;  }Graph;  void init(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void add(Graph \*G, int u, int v, int w, int d){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->edge[G->m].d = d;  G->m++;  }  int parent[MAX];  int findroot(int u){  while(parent[u] != u)  u = parent[u];  return u;  }  void Kruskal(Graph \*G, Graph \*T){  int u, v, w, d;  for(int i = 0; i < G->m; i++)  for(int j = i + 1; j < G->m; j++)  if(G->edge[i].w \* G->edge[i].d > G->edge[j].w \* G->edge[j].d){  Edge temp = G->edge[i];  G->edge[i] = G->edge[j];  G->edge[j] = temp;  }  init(T, G->n);  for(u = 1; u <= G->n; u++)  parent[u] = u;  int sum\_w = 0;  for(int e = 0; e < G->m; e++){  u = G->edge[e].u;  v = G->edge[e].v;  w = G->edge[e].w;  d = G->edge[e].d;  int root\_u = findroot(u);  int root\_v = findroot(v);  if(root\_u != root\_v){  add(T, u, v, w, d);  parent[root\_v] = root\_u;  sum\_w += w;  }  }  }  int main(){  Graph G, T;  int n, m, u, v, w, d, e;  scanf("%d%d",&n,&m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d%d",&u,&v,&w,&d);  add(&G, u, v, w, d);  }  Kruskal(&G, &T);  int cp = 0;  for(int i = 0; i < T.m; i++){  cp += T.edge[i].w \* T.edge[i].d;  }  printf("%d", cp);  } |
| Q6.  #include<stdio.h>  #define MAX 50  #define oo 9999  #define NO\_EDGE 0  typedef struct{  int A[MAX][MAX], n, m;  }Graph;  void init(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] = NO\_EDGE;  }  }  }  void add(Graph \*G, int u, int v, int w){  G->A[u][v] += w;  G->A[v][u] += w;  G->m++;  }  int pi[MAX];  int p[MAX];  int mark[MAX];  int Prim(Graph \*G, Graph \*T, int s){  int u, v, i, x;  for(u = 1; u <= G->n; u++){  pi[u] = oo;  p[u] = -1;  mark[u] = 0;  }  pi[s] = 0;  for(i = 1; i < G->n; i++){  int min\_dist = oo;  for(x = 1; x <= G->n; x++){  if(!mark[x] && pi[x] < min\_dist){  min\_dist = pi[x];  u = x;  }  }  mark[u] = 1;  for(v = 1; v <= G->n; v++){  if(!mark[v] && G->A[u][v] != NO\_EDGE && pi[v] > G->A[u][v]){  pi[v] = G->A[u][v];  p[v] = u;  }  }  }  init(T, G->n);  int sum\_w = 0;  for(u = 1; u <= G->n; u++){  if(p[u] != -1){  int w = G->A[p[u]][u];  add(T, p[u], u, w);  sum\_w += w;  }  }  return sum\_w;  }  int main(){  Graph G, T;  int n, m, u, v, w, e;  scanf("%d%d",&n, &m);  init(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d", &u, &v, &w);  add(&G, u, v, w);  }  init(&T, n);  int sum\_w = 0;  sum\_w = Prim(&G, &T, 1);  printf("%d\n", sum\_w);  } |

1. **Thuật toán tìm cây khung từ đồ thị có hướng Chuliu - Edmonds (k có trong giáo trình)**

=> Lấy test case trong bt lý thuyết r chạy code so đáp án.

|  |
| --- |
| #include<stdio.h>  #define MAXN 100  #define MAXM 500  #define MAXIT 10  #define oo 99999  typedef struct{  int u, v, w, link;  }Edge;  typedef struct{  int n, m;  Edge edge[MAXM];  }Graph;  typedef struct{  int n;  int parent[MAXN];  int weight[MAXN];  int link[MAXN];  }Tree;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void init\_tree(Tree \*T, int n){  T->n = n;  int i;  for(i = 1; i <= n; i++){  T->parent[i] = -1;  T->weight[i] = oo;  T->link[i] = -1;  }  }  void add\_edge(Graph \*G, int u, int v, int w, int link){  G->edge[G->m].u = u;  G->edge[G->m].v = v;  G->edge[G->m].w = w;  G->edge[G->m].link = link;  G->m++;  }  void buildH(Graph \*G, int root, Tree \*H){  init\_tree(H, G->n);  int e;  for(e = 0; e < G->m; e++){  int u = G->edge[e].u;  int v = G->edge[e].v;  int w = G->edge[e].w;  int link = G->edge[e].link;  if(w < H->weight[v]){  H->parent[v] = u;  H->weight[v] = w;  H->link[v] = link;  }  }  H->parent[root] = -1;  H->weight[root] = 0;  }  int id[MAXN];  int color[MAXN];  int find\_cycles(Tree \*H, int root){  int i, u, no = 0;  for(i = 1; i <= H->n; i++){  id[i] = -1;  color[i] = -1;  }  for(i = 1; i <= H->n; i++){  int u = i;  while(u != root && id[u] == -1 && color[u] != i){  color[u] = i;  u = H->parent[u];  }  if(color[u] == i){  no++;  int v = H->parent[u];  while(v != u){  id[v] = no;  v = H->parent[v];  }  id[u] = no;  }  }  return no;  }  void contract(Graph \*G, Tree \*H, int no, Graph \*G1){  init\_graph(G1, no);  int e;  for(e = 0; e < G->m; e++){  int u = G->edge[e].u;  int v = G->edge[e].v;  int w = G->edge[e].w;  if(id[u] != id[v])  add\_edge(G1, id[u], id[v], w - H->weight[v], e);  }  }  void expand(Tree \*H, Graph \*G1, Tree \*H1){  int i;  for(i = 1; i <= H->n; i++)  if(H->parent[i] != -1){  Edge pe = G1->edge[H->link[i]];  H1->parent[pe.v] = pe.u;  H1->weight[pe.v] += H->weight[i];  H1->link[pe.v] = pe.link;  }  }  void ChuLiu(Graph \*G0, int s, Tree \*T){  Graph G[MAXIT];  Tree H[MAXIT];  int i, e;  int t = 0;  int root = s;  G[0] = \*G0;  while(1){  buildH(&G[t], root, &H[t]);  int no = find\_cycles(&H[t], root);  if(no == 0) break;  for(i = 1; i <= H[t].n; i++){  if(id[i] == -1)  id[i] = ++no;  }  contract(&G[t], &H[t], no, &G[t+1]);  root = id[root];  t++;  }  int k;  for(k = t; k > 0; k--)  expand(&H[k], &G[k-1], &H[k-1]);  \*T = H[0];  }  int main(){  Graph G;  int n, m, i, e, u, v, w;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&w);  add\_edge(&G, u, v, w, -1);  }  printf("\n");  Tree T;  ChuLiu(&G, 1, &T);  for(i = 1; i <= T.n; i++)  if(T.parent[i] != -1)  printf("%d -> %d: %d\n", T.parent[i], i, T.weight[i]);  return 0; //kkk  }  Link drive: https://drive.google.com/file/d/1HRXyA3R4thDvqOYgqu3qq9sTX5dzoPTt/view |

1. **BT5.4. Tìm luồng cực đại trong mạng**

|  |
| --- |
| #include<stdio.h>  #define MAX 50  #define NO\_EDGE 0  #define oo 9999  typedef struct{  int dir, p, sigma;  }Label;  Label labels[MAX];  typedef struct{  int C[MAX][MAX], F[MAX][MAX], n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  void init\_flow(Graph \*G){  for(int u = 1; u <= G->n; u++)  for(int v = 1; v <= G->n; v++)  G->F[u][v] = NO\_EDGE;  }  typedef struct{  int hangdoi[MAX], front, rear;  }Queue;  void make\_null\_Queue(Queue \*Q){  Q->front = 0;  Q->rear = -1;  }  void enQueue(Queue \*Q, int u){  Q->hangdoi[++Q->rear] = u;  }  int front(Queue \*Q){  return Q->hangdoi[Q->front];  }  void deQueue(Queue \*Q){  Q->front++;  }  int emptyQueue(Queue \*Q){  return Q->front > Q->rear;  }  int min(int a, int b){  return a < b ? a : b;  }  int FordFulkerson(Graph \*G, int s, int t){  init\_flow(G);  int u, v;  int max\_flow = 0;  Queue Q;  do{  for(u = 1; u <= G->n; u++)  labels[u].dir = 0;  labels[s].dir = 0;  labels[s].p = s;  labels[s].sigma = oo;  make\_null\_Queue(&Q);  enQueue(&Q, s);  int found = 0;  while(!emptyQueue(&Q)){  int u = front(&Q); deQueue(&Q);  for(v = 1; v <= G->n; v++){  if(G->C[u][v] != NO\_EDGE && labels[v].dir == 0 && G->F[u][v] < G->C[u][v]){  labels[v].dir = +1;  labels[v].p = u;  labels[v].sigma = min(labels[u].sigma, G->C[u][v] - G->F[u][v]);  enQueue(&Q, v);  }  }  for(int x = 1; x <= G->n; x++){  if(G->C[x][u] != NO\_EDGE && labels[x].dir == 0 && G->F[x][u] > 0){  labels[x].dir = -1;  labels[x].p = u;  labels[x].sigma = min(labels[u].sigma, G->F[x][u]);  enQueue(&Q, x);  }  }  if(labels[t].dir != 0){  found = 1;  break;  }  }  if(found == 1){  int sigma = labels[t].sigma;  u = t;  while(u != s){  int p = labels[u].p;  if(labels[u].dir > 0)  G->F[p][u] += sigma;  else G->F[u][p] -= sigma;  u = p;  }  max\_flow += sigma;  }  else break;  }while(1);  return max\_flow;  }  int main(){  Graph G;  int n, m, u, v, e, c;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d",&u,&v,&c);  G.C[u][v] = c;  }  int max\_flow = FordFulkerson(&G, 1, n);  printf("Max flow: %d\n", max\_flow);  printf("S: ");  for(u = 1; u <= n; u++)  if(labels[u].dir != 0)  printf("%d ", u);  printf("\n");  printf("T: ");  for(u = 1; u <= n; u++)  if(labels[u].dir == 0)  printf("%d ", u);  return 0;  } |

1. **BT5.5. Kiểm tra luồng hợp lệ:**

|  |
| --- |
| #include<stdio.h>  #define MAX 50  #define NO\_EDGE 0  #define oo 9999  typedef struct{  int dir, p, sigma;  }Label;  Label labels[MAX];  typedef struct{  int C[MAX][MAX], F[MAX][MAX], n, m;  }Graph;  void init\_graph(Graph \*G, int n){  G->n = n;  G->m = 0;  }  int kiemtraluong(Graph \*G, int n, int s, int t){  for (int u = 1; u <= G->n; u++) {  for (int v = 1; v <= G->n; v++) {  if (G->F[u][v] > G->C[u][v]) {  return 0;  }  }  }  int out\_s = 0, in\_t = 0;  for (int u = 1; u <= G->n; u++) {  out\_s += G->F[s][u];  in\_t += G->F[u][t];  }  if (out\_s != in\_t) {  return 0;  }  for (int u = 2; u <= n-1; u++) {  int in\_u = 0, out\_u = 0;  for (int v = 1; v <= n; v++) {  in\_u += G->F[v][u];  out\_u += G->F[u][v];  }  if (in\_u != out\_u) {  return 0;  }  }  return 1;  }  int main(){  Graph G;  int n, m, u, v, e, c, f;  scanf("%d%d",&n,&m);  init\_graph(&G, n);  for(e = 0; e < m; e++){  scanf("%d%d%d%d",&u,&v,&c,&f);  G.C[u][v] = c;  G.F[u][v] = f;  }  if(kiemtraluong(&G, n, 1, G.n))  printf("YES");  else printf("NO");  return 0;  } |

1. ***Các bản cài đặt List, Stack, Queue*:**
2. **List:**

|  |
| --- |
| typedef struct{  int danhsach[100], size;  }List;  void make\_null\_L(List \*L){   1. >size = 0;   }  void push(List \*L, int u){  L->danhsach[L->size++] = u;  }  int element\_at(List \*L, int i){  return L->danhsach[i - 1];  }  int empty\_L(List \*L){  return L->size == 0;  }  void copy\_list(List \*L1, List \*L2){ //Copy từ ds L2 qua L1  make\_null\_L(L1); // Kh có dòng này ra màn hình đen  for(int u = 1; u <= L2->size; u++)  push(L1, element\_at(L2, u);  } |

1. **Stack:**

|  |
| --- |
| typedef struct{  int nganxep[MAX\_N];  int top\_idx;  }Stack;  void make\_null(Stack \*S){  S->top\_idx = 0;  }  void push\_back(Stack \*S, int u){  S->nganxep[S->top\_idx++] = u;  }  int top(Stack \*S){  return S->nganxep[S->top\_idx - 1];  }  void pop(Stack \*S){  S->top\_idx--;  }  int empty(Stack \*S){  return S->top\_idx == 0;  } |

1. **Queue:**

|  |
| --- |
| #define MAX\_SIZE 50  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;  } |