**THỰC HÀNH BUỔI 4**

1. **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");  } |