

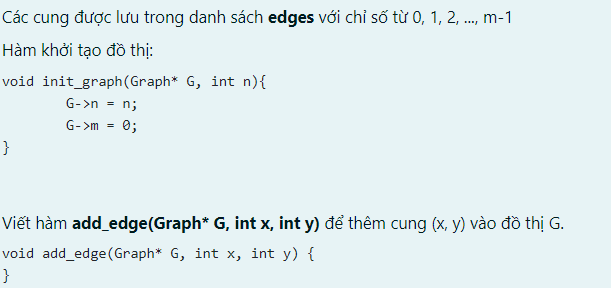
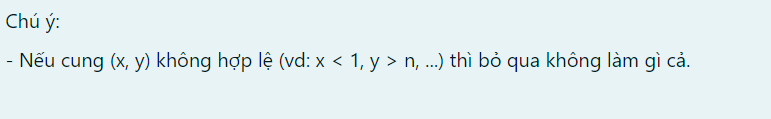
void add\_edge(Graph\* G, int x, int y) {

G->edges[G->m].x=x;

G->edges[G->m].y=y;

G->m++;

}



void add\_edge(Graph\* G, int x, int y) {

if( x>0 && x<=G->n && y>0 && y<=G->n){

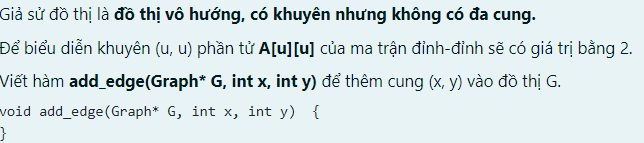
G->edges[G->m].x=x;

G->edges[G->m].y=y;

G->m++;

}

}



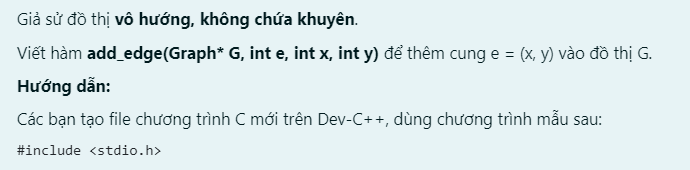
void add\_edge(Graph\* G, int x, int y) {

G->A[x][y]+=1;

G->A[y][x]+=1;

G->m++;

}

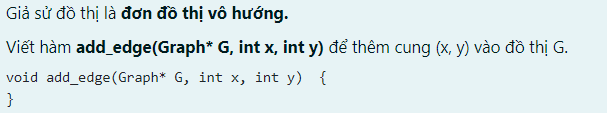


void add\_edge(Graph \*G,int e,int x,int y){

G->A[x][e]=1;

G->A[y][e]=1;

}



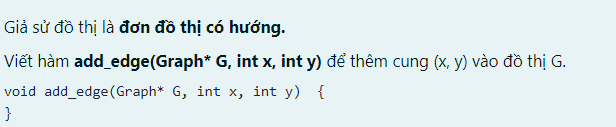
void add\_edge(Graph\* G, int x, int y) {

G->A[x][y]=1;

G->A[y][x]=1;

G->m++;

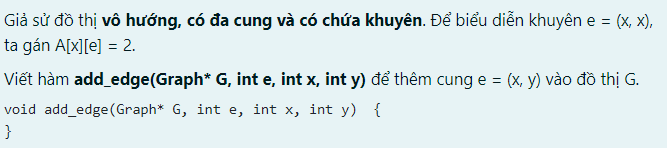
}



void add\_edge(Graph\* G, int x, int y) {

G->A[x][y]=1;

G->m++;

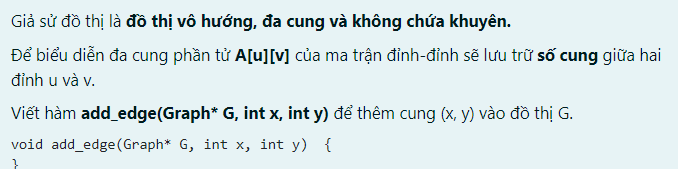
}

void add\_edge(Graph\* G, int e, int x, int y) {

G->A[x][e]+=1;

G->A[y][e]+=1;

G->m++;

}

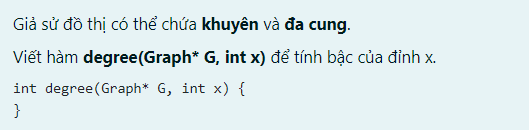
void add\_edge(Graph\* G, int x, int y) {

G->A[x][y]+=1;

G->A[y][x]+=1;

G->m++;

}



int degree(Graph\* G, int x) {

int i,deg=0;

for(i=0;i<=G->m;i++){

if( (G->edges[i].x==x) || (G->edges[i].y==x)){

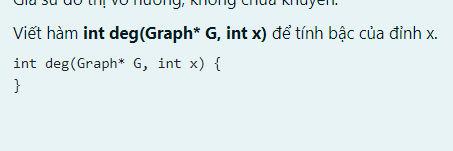
deg++;

}

}

return deg;

}



int deg(Graph \*G,int x){

int i,deg=0;

for(i=1;i<=G->m;i++){

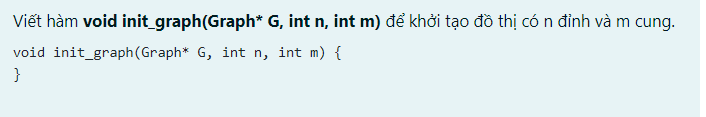
if(G->A[x][i]!=0){

deg++;

}

}return deg;

}



void init\_graph(Graph\* G, int n, int m) {

int i,j;

G->n=n;

G->m=m;

for(i=1;i<=G->n;i++){

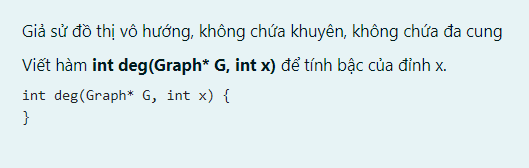
for(j=1;j<=G->n;j++){

G->A[i][j]=0;

}

}

}



int deg(Graph\* G, int x) {

int y,deg=0;

for(y=1;y<=G->n;y++){

if(G->A[x][y]!=0){

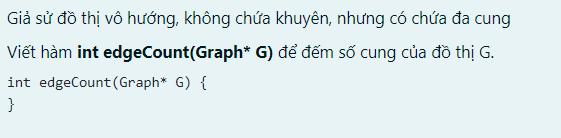
deg++;

}

}

return deg;

}



int edgeCount(Graph \*G){

int u,v, dem=0;

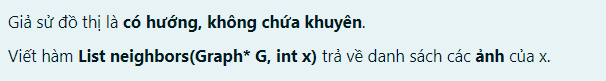
for (u=1;u<=G->n;u++)

for(v=1;v<=u;v++)

if(G->A[u][v]!=0)

dem+=G->A[u][v];

return dem;

}

List neighbors(Graph\* G, int x){

List l;

for(int i=1;i<=G->n;i++){

if(G->A[x][i]!=0){

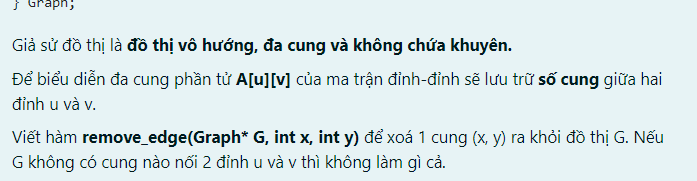
push\_back(&l,i);

}

}

return l;

}



void remove\_edge(Graph\* G, int x, int y) {

if(G->A[x][y]==0)

return ;

G->A[x][y]-=1;

G->A[y][x]-=1;

G->m--;

}