Exc3

A)

```
  Grafo* cria_Grafo(int nro_vertices, int grau_max, int
eh_ponderado){

 Grafo *gr; //cria um tipo grafo
   gr = (Grafo*) malloc(sizeof(struct grafo));// alloca ele
dinamicamente
   if(gr != NULL){
       int i;
       gr->nro_vertices = nro_vertices;
       gr->grau_max = grau_max;
       gr->eh_ponderado = (eh_ponderado != 0)?1:0; //If ternário
perguntando se é ponderado
       gr->grau = (int*) calloc(nro_vertices,sizeof(int));
       // instancia grau usando uma função calloc que é igual a
malloc porém zerando os
       // numeros

       /*Instancia a estrutura arestas que tem os nós conectados,
a partir de
       um malloc de ponteiro duplo que cria a coluna inicial e pra
cada elemento da
       coluna é criada uma lista dentro dela */
       gr->arestas = (int**) malloc(nro_vertices * sizeof(int*));
       for(i=0; i<nro_vertices; i++)
           gr->arestas[i] = (int*) malloc(grau_max * sizeof(int));

       /* Verifica  se é ponderado e se sim, cria um igual ao de
cima
        só que pro peso, se não usa só as arestas */
       if(gr->eh_ponderado){
           gr->pesos = (float**) malloc(nro_vertices *
sizeof(float*));
           for(i=0; i<nro_vertices; i++)
               gr->pesos[i] = (float*) malloc(grau_max *
sizeof(float));
       }

   }
   return gr;
}
```

B)

```
 void libera_Grafo(Grafo* gr){
  if(gr != NULL){ //Verifica se o grafo é válido
       int i;
       for(i=0; i<gr->nro_vertices; i++) //Libera a Matriz de
arestas linha por linha
           free(gr->arestas[i]);
       free(gr->arestas); //Libera o vetor principal arestas

       if(gr->eh_ponderado){ //Verifica se o grafo é ponderado e
se sim, libera os pesos
           for(i=0; i<gr->nro_vertices; i++)
               free(gr->pesos[i]);
           free(gr->pesos);
       }
       free(gr->grau); //Libera o vetor grau dos vértices
       free(gr); // Por fim, libera a estrutura principal
   }
}
```

C)

```
int insereAresta(Grafo* gr, int orig, int dest, int eh_digrafo,
float peso){



//Verifica se o grafo é valido
   if(gr == NULL)
       return 0;
   if(orig < 0 || orig >= gr->nro_vertices)
       return 0;
   if(dest < 0 || dest >= gr->nro_vertices)
       return 0;

   /* Pega arestas na posição de Origem(item mais alta e a
esquerda, geralmente 1)
    e acessa o grau da origem atualizando-o para destino( a aresta
que será inserida)
    Basicamente fazendo a conexão da origem com o destino */
   gr->arestas[orig][gr->grau[orig]] = dest;
   if(gr->eh_ponderado) //Verifica se é ponderado
       gr->pesos[orig][gr->grau[orig]] = peso;// e caso seja, põe
o peso nessa conexão
   gr->grau[orig]++; //aumenta grau na posição de origem em um( já
que fez uma nova conexão)

   if(eh_digrafo == 0)// Por fim verifica se é digrafo( se tem
direcionamento)
       insereAresta(gr,dest,orig,1,peso); /* se ele não é digrafo
   então precisamos passar tudo pros dois lado , ent ele chama a
função mas inverte
   a origem com o destino */
   return 1;
}
```

D)

```
 int removeAresta(Grafo* gr, int orig, int dest, int eh_digrafo){
  //Verifica se o grafo é válido
   if(gr == NULL)
       return 0;
   if(orig < 0 || orig >= gr->nro_vertices)
       return 0;
   if(dest < 0 || dest >= gr->nro_vertices)
       return 0;

   int i = 0;
   while(i<gr->grau[orig] && gr->arestas[orig][i] != dest)
/*Procura a posição da aresta a
posição da aresta a ser removida pela lista de vizinhos da origem
até achar destino */
       i++;
   if(i == gr->grau[orig])//elemento nao encontrado, se não
encontrou retorna a 0
       return 0;
   gr->grau[orig]--; // Diminui o grau de origem
   gr->arestas[orig][i] =
gr->arestas[orig][gr->grau[orig]];//Substitui o elemento da
posição
   // i pelo ultimo da lista, mas como já está decrementado, ele
acha o antigo ultimo elemento
   if(gr->eh_ponderado)// Se for ponderado , remove o peso
correspondente
       gr->pesos[orig][i] = gr->pesos[orig][gr->grau[orig]];
   if(eh_digrafo == 0)// Se não for digrafo(direcionado), chama
recursivamente para deletar
       //o outro lado, mas passa 1 como ehdigrafo para não chamar
infinitamente
       removeAresta(gr,dest,orig,1);
   return 1;
}
```