Grafos

```
//Definição do tipo Grafo
typedef struct grafo{
   int eh_ponderado;
   int nro_vertices;
   int grau_max;
   int** arestas;
   float** pesos;
   int* grau;
}Grafos;

Grafo* cria_Grafo(int nro_vertices, int grau_max, int
eh_ponderado){
   Grafo *gr;
   gr = (Grafo*) malloc(sizeof(struct grafo));
   if(gr != NULL){
      int i;
      gr->nro_vertices = nro_vertices;
      gr->grau_max = grau_max;
      gr->eh_ponderado = (eh_ponderado != 0)?1:0;
      gr->grau = (int*) calloc(nro_vertices,sizeof(int));

      gr->arestas = (int**) malloc(nro_vertices *
sizeof(int*));
      for(i=0; i<nro_vertices; i++)
         gr->arestas[i] = (int*) malloc(grau_max *
sizeof(int));

      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;
}

int insereAresta(Grafo* gr, int orig, int dest, int
eh_digrafo, float peso){
   if(gr == NULL)
      return 0;
   if(orig < 0 || orig >= gr->nro_vertices)
      return 0;
   if(dest < 0 || dest >= gr->nro_vertices)
      return 0;

   gr->arestas[orig][gr->grau[orig]] = dest;
   if(gr->eh_ponderado)
      gr->pesos[orig][gr->grau[orig]] = peso;

   gr->grau[orig]++;

   if(eh_digrafo == 0)
      insereAresta(gr,dest,orig,1,peso);
   return 1;
}
```

Busca em Profundidade

```
void buscaProfundidade(Grafo *gr, int ini, int
*visitado, int cont){
   int i;
   visitado[ini] = cont;
   for(i=0; i<gr->grau[ini]; i++){
      if(!visitado[gr->arestas[ini][i]])
         buscaProfundidade(gr,gr-
>arestas[ini][i],visitado,cont+1);
   }
}
void buscaProfundidade_Grafo(Grafo *gr, int ini, int
*visitado){
   int i, cont = 1;
   for(i=0; i<gr->nro_vertices; i++)
      visitado[i] = 0;
   buscaProfundidade(gr,ini,visitado,cont);
}
```

Busca em Largura

```
void buscaLargura_Grafo(Grafo *gr, int ini, int
*visitado){
   int i, vert, NV, cont = 1;
   int *fila, IF = 0, FF = 0;
   for(i=0; i<gr->nro_vertices; i++)
      visitado[i] = 0;
   NV = gr->nro_vertices;
   fila = (int*) malloc(NV * sizeof(int));
   FF++;
   fila[FF] = ini;
   visitado[ini] = cont;
   while(IF != FF){
      IF = (IF + 1) % NV;
      vert = fila[IF];
      cont++;
      for(i=0; i<gr->grau[vert]; i++){
         if(!visitado[gr->arestas[vert][i]]){
            FF = (FF + 1) % NV;
            fila[FF] = gr->arestas[vert][i];
            visitado[gr->arestas[vert][i]] = cont;
         }
      }
   }
   free(fila);
}
```

Busca pelo menor Caminho

```
void menorCaminho_Grafo(Grafo *gr, int ini, int *ant,
float *dist){
   int i, cont, NV, ind, *visitado, vert;
   cont = NV = gr->nro_vertices;
   visitado = (int*) malloc(NV * sizeof(int));
   for(i=0; i < NV; i++){
      ant[i] = -1;
      dist[i] = -1;
      visitado[i] = 0;
   }
   dist[ini] = 0;
   while(cont > 0){
      vert = procuraMenorDistancia(dist, visitado, NV);
      if(vert == -1)
         break;

      visitado[vert] = 1;
      cont--;
      for(i=0; i<gr->grau[vert]; i++){
         ind = gr->arestas[vert][i];
         if(dist[ind] < 0){
           dist[ind] = dist[vert] + 1;
           ant[ind] = vert;
         }else{
            if(dist[ind] > dist[vert] + 1){
               dist[ind] = dist[vert] + 1;
               ant[ind] = vert;
            }
         }
      }
   }

   free(visitado);
}

int procuraMenorDistancia(float *dist, int *visitado,
int NV){
   int i, menor = -1, primeiro = 1;
   for(i=0; i < NV; i++){
      if(dist[i] >= 0 && visitado[i] == 0){
         if(primeiro){
            menor = i;
            primeiro = 0;
         }else{
            if(dist[menor] > dist[i])
               menor = i;
         }
      }
   }
   return menor;
}
```