Exemplo de Exercícios para a Prova 2 – ACH2024 – Algoritmos e Estruturas de Dados II

Questão 1. A função de impressão de grafos vista em aula foi chamada para imprimir um grafo direcionado e ponderado representado por listas de adjacências. O resultado do que foi impresso pode ser visto a seguir. Desenhe o **grafo direcionado e ponderado** correspondente.

```
Imprimindo grafo (vertices: 6; arestas: 10).
[ 0] -> 0 (1.00) -> 3 (3.50)
[ 1] -> 0 (6.00) -> 4 (5.00)
[ 2] -> 1 (2.50) -> 3 (2.00) -> 4 (6.00)
[ 3] -> 5 (4.00)
[ 4] -> 5 (3.00)
[ 5] -> 2 (4.50)
```

Desenhe o grafo aqui:

Questão 2. Dado o seguinte grafo **ponderado e não direcionado**, representado pela matriz de adjacências com pesos apresentada a seguir, desenhe a **árvore geradora de custo mínimo** para esse grafo.

	v0	v1	v2	v3
v0	-	5.00	12.00	-
v1	5.00	-	9.00	10.00
v2	12.00	9.00	-	4.00
v3	-	10.00	4.00	-

Desenhe a árvore geradora de custo mínimo aqui:

Questão 3. Considerando as diferentes estratégias de alocação de blocos no disco e organização dos registros pelos blocos estudadas ao longo do semestre. Qual a complexidade assintótica das operações de inserção, exclusão e busca, no pior caso:

	Inserção	**Exclusão**	**Busca**
- Sequencial não ordenado (*heap files*)			
- Sequencial ordenado (*sorted files*)			
- Por listas ligadas (ordenação pelas chaves)			
- Indexada			
- Arvores B / B+			
- Hashing – Endereçamento Fechado			
- Hashing – Endereçamento Aberto			

Questão 4. Considerando a representação de grafos **não ponderados e não direcionados** usando **matrizes de adjacências booleanas**, com base na estrutura apresentada a seguir (que é a mesma que foi vista em aula), implemente as duas funções solicitadas.

```
typedef struct {
  int numVertices;
  int numArestas;
  bool** matriz;
} Grafo;
```

a) verticeMaiorGrau: função que recebe o endereço de um grafo (g) e retorna o número/identificador do vértice que tem o maior grau no grafo (se mais de um vértice tiver o maior grau, sua função deve retornar o menor deles [aquele com o menor identificador]). Você pode considerar que g contém um endereço válido para um grafo. **Não use nenhuma função auxiliar.**

```
int verticeMaiorGrau(Grafo* g){









}
```

b) vizinhanca2: função que recebe o endereço de um grafo (g) e o identificador de um vértices (u) e retorna o número total de vértices que são vizinhos de u ou vizinhos dos vizinhos de u (isto é o número de vizinhos de u, mais o número de vizinhos dos vizinhos de u [cuidado para não contar um mesmo vértice duas vezes e você não deve incluir u na contagem). Você pode considerar que g contém um endereço válido para um grafo.

```
int vizinhanca2(Grafo* g, int u){









}
```

Questão 5. O algoritmo de Dijkstra, que calcula o arranjo de distância e de predecessores de um vértice para os demais vértices de um grafo, foi executado para um dado grafo, a partir do vértice 0 (zero) e produziu os seguintes arranjos de distância e de predecessores:

```
Exibindo arranjo de distâncias.
```

v0	v1	v2	v3	v4	v5
0.00	3.50	1.00	3.00	7.00	7.00

```
Exibindo arranjo de predecessores.
```

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Com base nesses arranjos, escreva qual será o caminho de menor custo (escreva a sequência de vértices na ordem em que serão percorridos) e o custo (ou distância) total para:

a) Sair do vértice **v0** e chegar no vértice **v4**:

Caminho: Distância:

b) Sair do vértice **v0** e chegar no vértice **v5**:

Caminho: Distância:

c) Sair do vértice **v0** e chegar no vértice **v3**:

Caminho: Distância:

Questão 6. O algoritmo de Floyd-Warshall, que calcula as distâncias (e os predecessores) entre todos os pares de vértices, foi executado para um dado grafo (ilustrado a seguir pela impressão do grafo no formato de listas de adjacências) e produziu as seguintes matrizes de distâncias e de predecessores:

```
Imprimindo grafo (vértices: 6; arestas: 13).
[ 0] ->  3 (10.00) ->  5 (76.00)
[ 1] ->  2 (21.00) ->  4 (14.00)
[ 2] ->  2 (73.00) ->  5 (44.00)
[ 3] ->  1 (57.00) ->  2 (70.00) ->  5 (78.00)
[ 4] ->  1 (16.00) ->  3 (55.00)
[ 5] ->  0 (55.00) ->  4 (47.00)
```

Exibindo matriz de distâncias.

	v0	v1	v2	v3	v4	v5
v0	0.00	67.00	80.00	10.00	81.00	76.00
v1	120.00	0.00	21.00	69.00	14.00	65.00
v2	99.00	107.00	0.00	109.00	91.00	44.00
v3	133.00	57.00	70.00	0.00	71.00	78.00
v4	136.00	16.00	37.00	55.00	0.00	81.00
v5	55.00	63.00	84.00	65.00	47.00	0.00

Exibindo matriz de predecessores.

	v0	v1	v2	v3	v4	v5
v0	0	3	3	0	1	0
v1	5	1	1	4	1	2
v2	5	4	2	0	5	2
v3	5	3	3	3	1	3
v4	5	4	1	4	4	2
v5	5	4	1	0	5	5

a) Qual é a menor distância entre o vértice v0 e o vértice v4? _____

b) Qual vértice está mais distante de v2? _____

c) Qual é o par de vértices mais distante (isto é, se você precisar sair de um vértice x e chegar a um vértice y, qual é o par x e y cujo caminho tem a maior distância)?

d) O grafo desta questão é cíclico ou acíclico? _____

e) O grafo desta questão é fortemente conexo? _____

f) Desenhe a Árvore de Caminhos de Menor Custo (ou de Custo Mínimo) do vértice v3 (isto é, aquela em que v3 é a raiz):	**f)** Desenhe a Árvore de Caminhos de Menor Custo (ou de Custo Mínimo) do vértice v4 (isto é, aquela em que v4 é a raiz):

Questão 7. Considerando os quatro "tipos" de arestas potencialmente identificadas durante a busca em profundidade em grafos direcionados (aresta de árvore, de retorno, de avanço e de cruzamento).
Que características definem cada uma destas arestas?
Aresta de árvore: (exemplo de resposta) é a aresta (x,y) pela qual, na busca em profundidade, se visita o vértice y pela primeira vez. As arestas de árvore são as arestas que estarão presentes nas Árvores de Busca em Profundidade.

Aresta de Retorno:

Aresta de Avanço:

Aresta de Cruzamento:

Questão 8. Em um grafo direcionado que possui arestas de retorno é possível realizar uma ordenação topológica?

Questão 9. Considerando os diferentes quatro tipos de arestas da questão anterior, qual (ou quais) destas arestas sempre estará(ão) presente(s) em um grafo direcionado **cíclico**?

Questão 10. Compare (em termos de vantagens e desvantagens) o uso de hashing considerando: Endereçamento Fechado e Endereçamento Aberto.

Questão 11. Estamos usando uma estrutura de hashing e, por azar, todas as n inserções que realizamos mapearam para a mesma posição inicial. Qual o custo computacional (ou complexidade) da n-ésima inserção:
a) Considerando o uso de Endereçamento Fechado:

b) Considerando o uso de Endereçamento Aberto com Sondagem Linear:

c) Considerando o uso de Endereçamento Aberto com Sondagem Quadrática:

Questão 12. Implemente a função de divisão de nó cheio em Árvores B, considere as definições abaixo (que são as mesmas que foram usadas nos códigos em aula).
A função a ser implementada tem a seguinte assinatura:
`void divideNoFilho(No* pai, int i, No* filho)`
Sendo *pai* o endereço do pai do nó que será dividido, *filho* o endereço do nó que será dividido e *i* a posição no arranjo *filhos* do nó pai em que está o ponteiro para o nó *filho*.

```
#define T 3
typedef int bool;
typedef int TipoChave;
typedef struct {
 TipoChave chave;
 // OUTROS CAMPOS
} Registro;

typedef struct auxNo{
 int numChaves;
 bool folha;
 Registro regs[2*T-1];
 struct auxNo* filhos[2*T];  // poderiamos alocar dinamicamente sob
demanda
} No;

typedef struct{
 No* raiz;
} ArvB;

void divideNoFilho(No* pai, int i, No* filho){









}
```

Questão 13. Desenhe a Árvore B com grau mínimo igual a 3 (T=3) resultante da inserção, em uma árvore inicialmente vazia, das chaves de 1 a 10, nessa ordem (em ordem crescente).

Questão 14. Dada a árvore resultante da questão anterior (após as dez inserções), realize as exclusões das chaves de 4 a 7, nessa ordem (primeiro a remoção da chave 4, depois da 5 e assim por diante) e desenhe a árvore final resultante.

Questão 15. Durante o semestre usamos uma implementação de Conjuntos Disjuntos para auxiliar na implementação do algoritmo de Kruskal. Nessa implementação, baseada em florestas, cada elemento tinha um ponteiro para o seu pai. Inicialmente cada elemento pertencia a um conjunto diferente: todos os ponteiros para os pais iniciavam com o valor NULL (árvores formadas por um único elemento) e eventualmente essas árvores eram unidas (a raiz de uma árvore apontava para a raiz de outra como seu pai). Para descobrir se dois elementos pertencem ao mesmo conjunto, verificamos se ambos possuem a mesma raiz (se estão na mesma árvore). Abaixo está a função que retorna o endereço da raiz de um dado elemento u.

```
PontC identificaConjunto(Conjunto* c, int u){
 if (!c || u<0 || u>=c->numElementos) return NULL;
 PontC atual = c->elementos[u];
 while(atual->pai)
   atual = atual->pai;
 return atual;
}
```

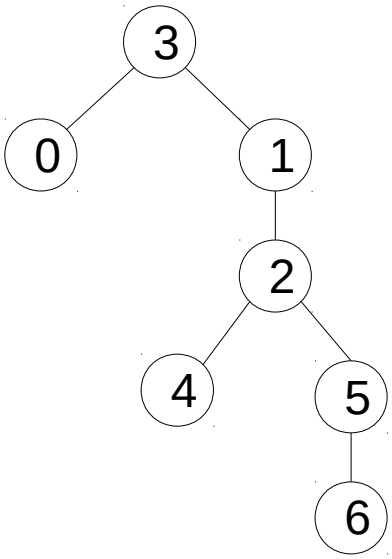
Nós otimizamos a função para, durante a busca, potencialmente diminuir a altura da árvore:

```
PontC identificaConjuntoOtim(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  PontC atual = c->elementos[u];
  while(atual->pai) {
     atual = atual->pai;
     c->elementos[u]->pai = atual;
  }
  return atual;
}
```

Já a seguinte otimização recursiva também objetia potencialmente diminuir a altura da árvore de outra forma:

```
PontC identificaConjuntoRec(Conjunto* c, int u){
  if (!c || u<0 || u>=c->numElementos) return NULL;
  if(c->elementos[u]->pai)
    c->elementos[u]->pai=identificaConjuntoRec(c,c->elementos[u]->pai->id);
  return c->elementos[u];
}
```

Dada a seguinte árvore, desenhe a árvore resultante ao se buscar pelo elemento 5, considerando cada uma das funções de otimização:

Árvore Original	**IdentificaConjuntoOtim**	**identificaConjuntoRec**
		

Questão 16. Pretendemos usar duas formas de hashing dinâmico para inserir o seguinte conjunto de registros na ordem em que aparecem a seguir (para esta prova, só nos interessa o valor da função hash associada ao registro [valor em binário e decimal] e o "nome do registro" que será dado por uma letra maiúscula). Considere que em cada bloco cabem apenas dois registros.

Valor binário	Valor Decimal	Registro
0001	1	A
0010	2	B
1001	9	C
0101	5	D
0110	6	E
1101	13	F
1111	15	G
1010	12	H

a) Desenhe como ficará o Hashing Dinâmico **Extensível**, após a inserção destes oito registros. Considere que ele é inicializado vazio com tamanho 2 (i=1).

b) Desenhe como ficará o Hashing Dinâmico **Linear**, após a inserção destes oito registros. Considere que ele é inicializado vazio com tamanho 2.