Nome: Juliano Leonardo Soares
Curso: Ciência da Computação

Respostas para os exercícios propostos devem ser submetidas em arquivo pdf por email para o prof.

Aulas 05 e 06 (a resposta para o exercício 8 será usada como critério de "presença" nestas aulas 05 e 06 e determinação da conceito final na disciplina)

(8) Trabalho individual: usar algoritmos de busca em espaços de estados para implementar a resolução do problema dos missionários e canibais (descrito na aula 05).

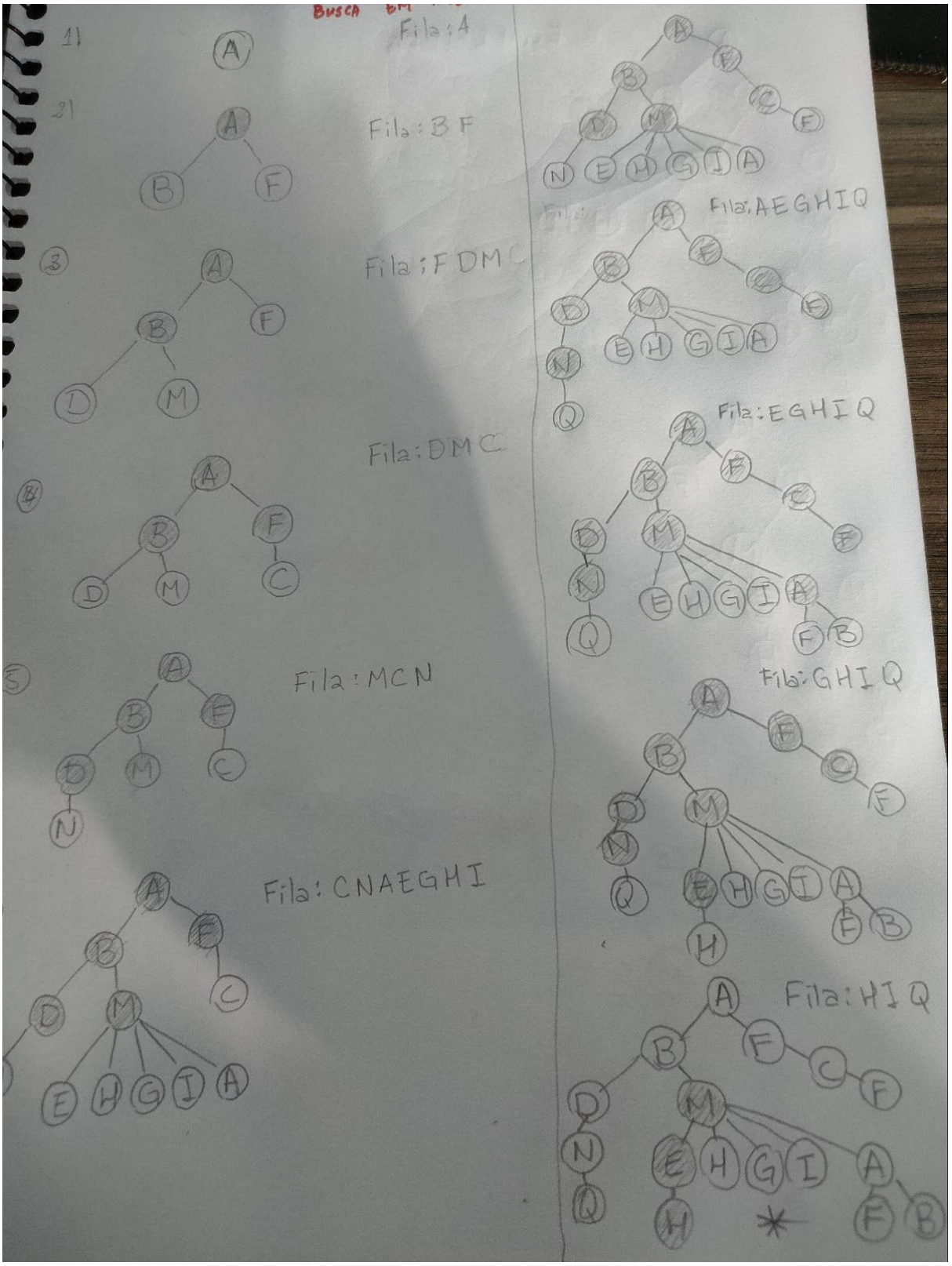
```
# situação inicial
# 0 - n missionarios l esq
# 1 - n canibais l esquerdo
# 2 - n missionarios l direito
# 3 - n canibais l direito
# 4 - Canoa esta onde 0 - esquerdo 1 - direito
estadoInicial=[3,3,0,0,0]

# operadores
# (1,0) - dupla referente primeira posição numero de missionario e segunda posição numero de canibais
# a canoa só atravessa de dois em dois
operadores = [(1,0),(1,1),(2,0),(0,2)]

borda = []
visitados =[]

def moveCanoa(estadoAtual, nMisionarios = 0, nCanibais = 0):
    if nMisionarios + nCanibais > 2:
        return
    if estadoAtual[-1] == 0: # posição final do estado lado esquerdo
        origemM = 0
        origemC = 1
        destinoM = 2
        destinoC = 3
    else:
        origemM = 2
        origemC = 3
        destinoM = 0
        destinoC = 1

    if estadoAtual[origemM] == 0 and estadoAtual[origemC] == 0:
        print("Todos já transportados")
        return
```

```
    estadoAtual[-1] = 1 - estadoAtual[-1]
    for i in range(min(nMisionarios, estadoAtual[origemM])):
        estadoAtual[origemM]-=1
        estadoAtual[destinoM]+=1

    for i in range(min(nCanibais, estadoAtual[origemC])):
        estadoAtual[origemC]-=1
        estadoAtual[destinoC]+=1

    return estadoAtual

def sucessores(estado):
    sucessores =[]
    for (i,j) in operadores:
        s = moveCanoa(estado[:], i, j)
        if s == None: continue
        if (s[0] < s[1] and s[0]>0) or (s[2]<s[3] and s[2]>0): continue
        if s in visitados: continue
        sucessores.append(s)
    return sucessores

sucessores(estadoInicial)

def adjNVisitado(elemento):
    l = sucessores(elemento)
    if len(l)>0:
        return l[0]
    else:
        return -1

def atingiu(estado):
    if estado[2] >= 3 and estado[3] >= 3:
        return True
    else:
        return False

def buscaProfundidade(estadoInicial):
    borda.append(estadoInicial)
    while len(borda) != 0:
        elemento = borda[len(borda)-1]
        if atingiu(elemento): break
        v = adjNVisitado(elemento)
        if v == -1:
            borda.pop()
        else:
            visitados.append(v)
```

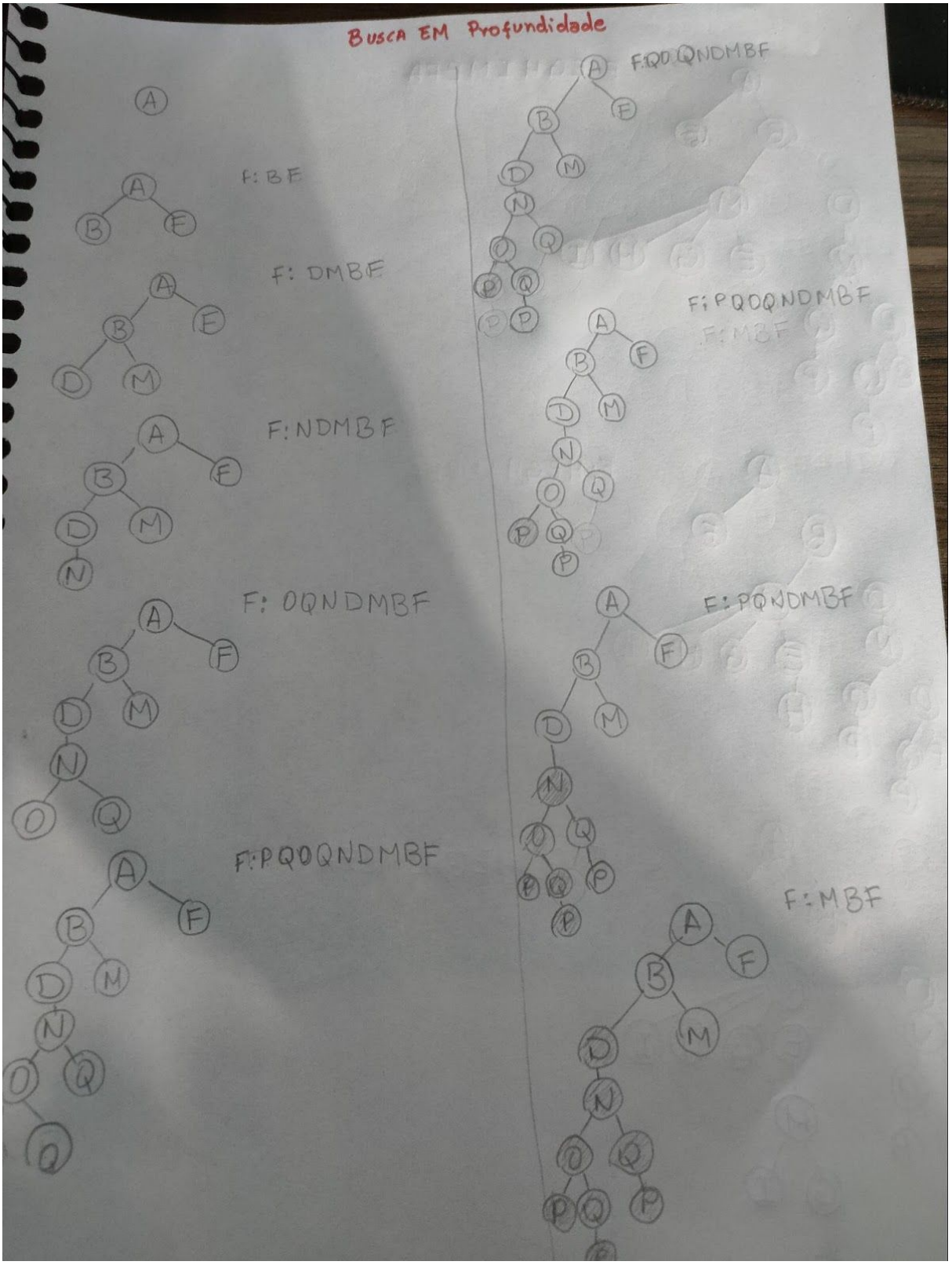
```
            borda.append(v)
    else:
        print("sem caminho")
    return borda

sol = buscaProfundidade(estadoInicial)

for i in range(1,len(sol)):
    destinoM = abs(sol[i][0] - sol[i-1][0])
    destinoC = abs(sol[i][1] - sol[i-1][1])
    canoa = sol[i][4] - sol[i-1][4]

    if canoa ==1:
        s = "-->"
    else:
        s = "<--"
    print(sol[i-1], "({},{},{})".format(destinoM,destinoC, s))
```

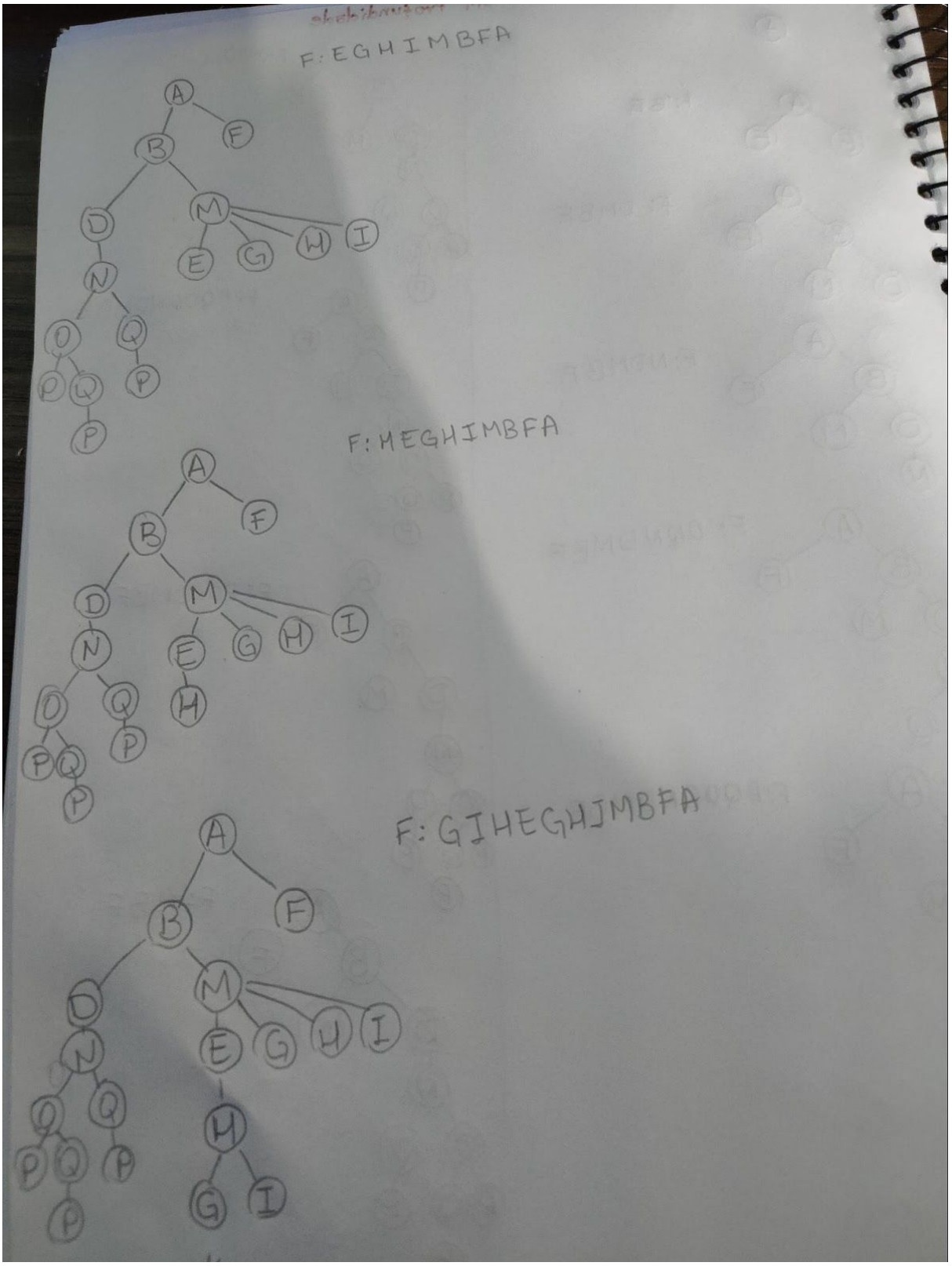
Aulas 07 e 08 (as respostas para os exercícios 9 e 10 serão usadas como critério de "presença" nestas aulas 07 e 08 e determinação do conceito final na disciplina)

(9) Detalhar (passo a passo) a execução dos algoritmos de busca em largura e profundidade no grafo descrito no final da aula 07


BUSCA
1)
Fila: A
2)
Fila: B F
3)
Fila: F D M C
4)
Fila: D M C
5)
Fila: M C N
Fila: C N A E G H I
Fila: A E G H I Q
Fila: E G H I Q
Fila: G H I Q
Fila: H I Q

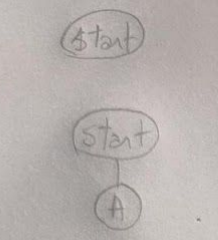
Busca em Profundidade





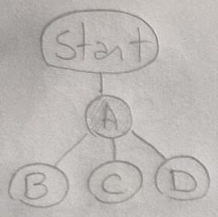
(10) Detalhar (passo a passo) e mostrar as árvores de busca que seriam geradas a partir da execução dos algoritmos de busca em largura, busca em profundidade e busca com aprofundamento iterativo no grafo descrito no final da aula 08

BUSCA EM largura

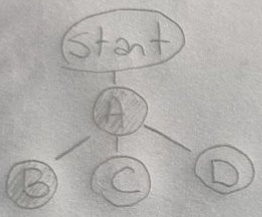


Fila: start

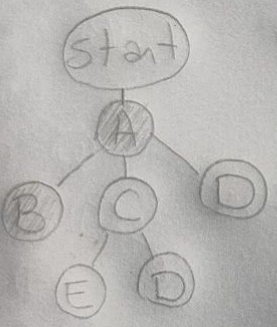
fila: A, A



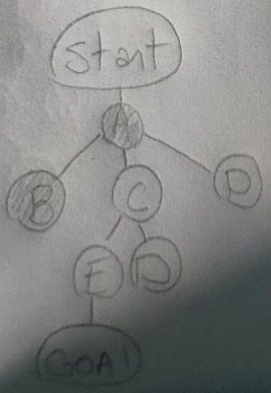
fila: B, C, D, A



f: C, D

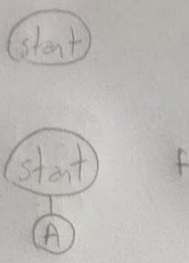


f: E, D, C, D

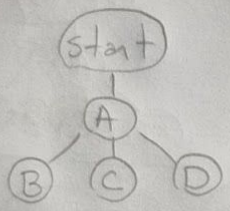


f: GOAL, E, D, C, D

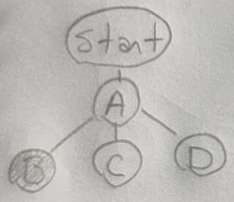
BuscaEmProfundidade



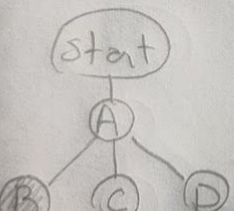
f: A



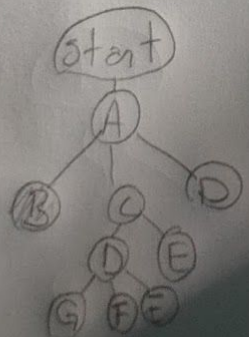
f: B, C, D, A



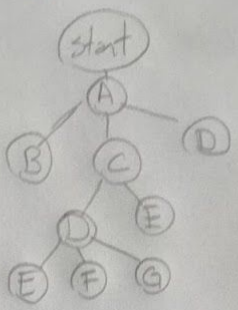
f: C, D, A

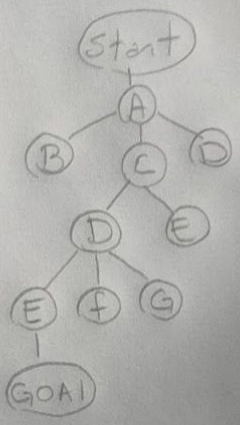


f: D, E, C, D, A

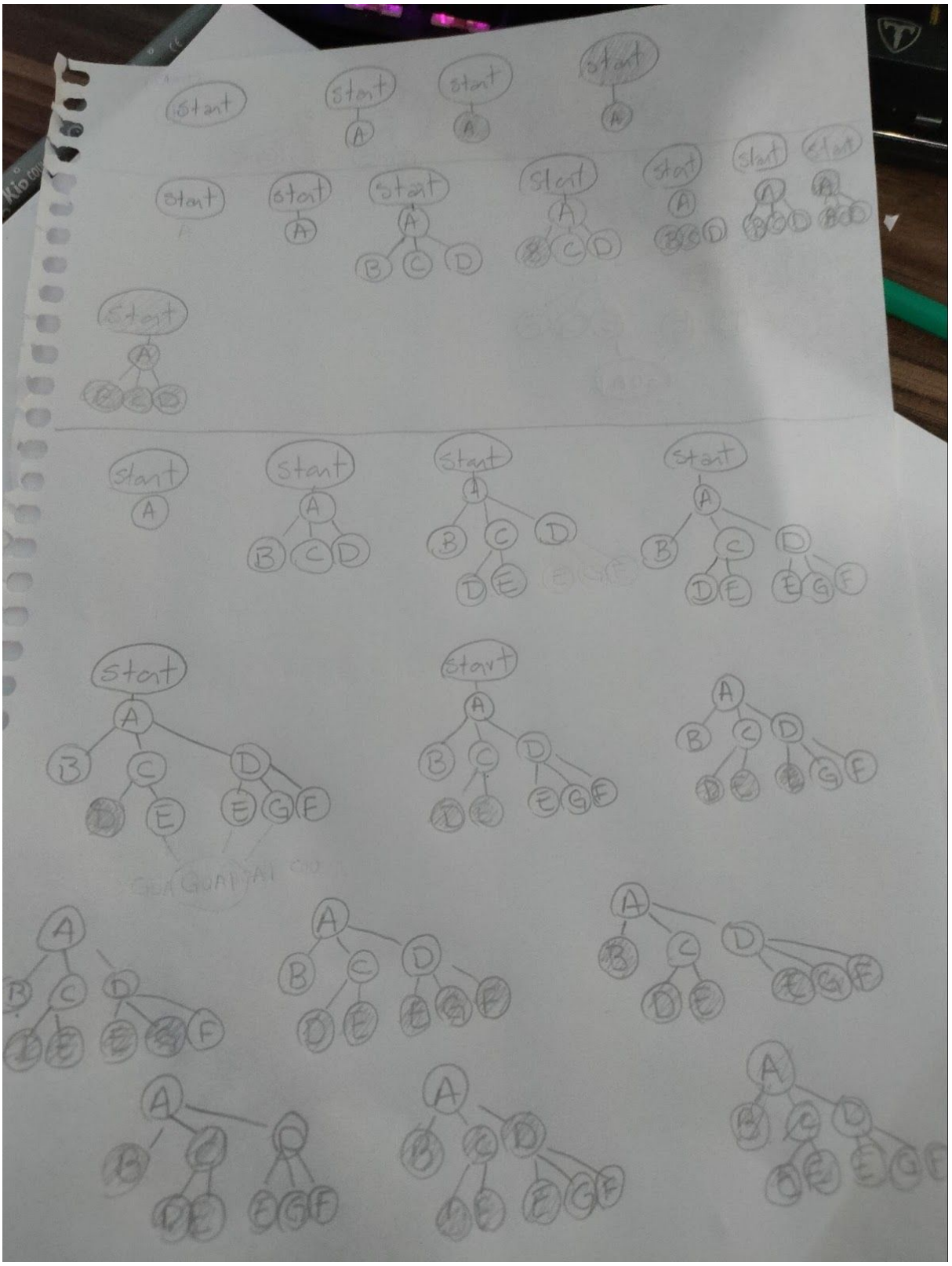


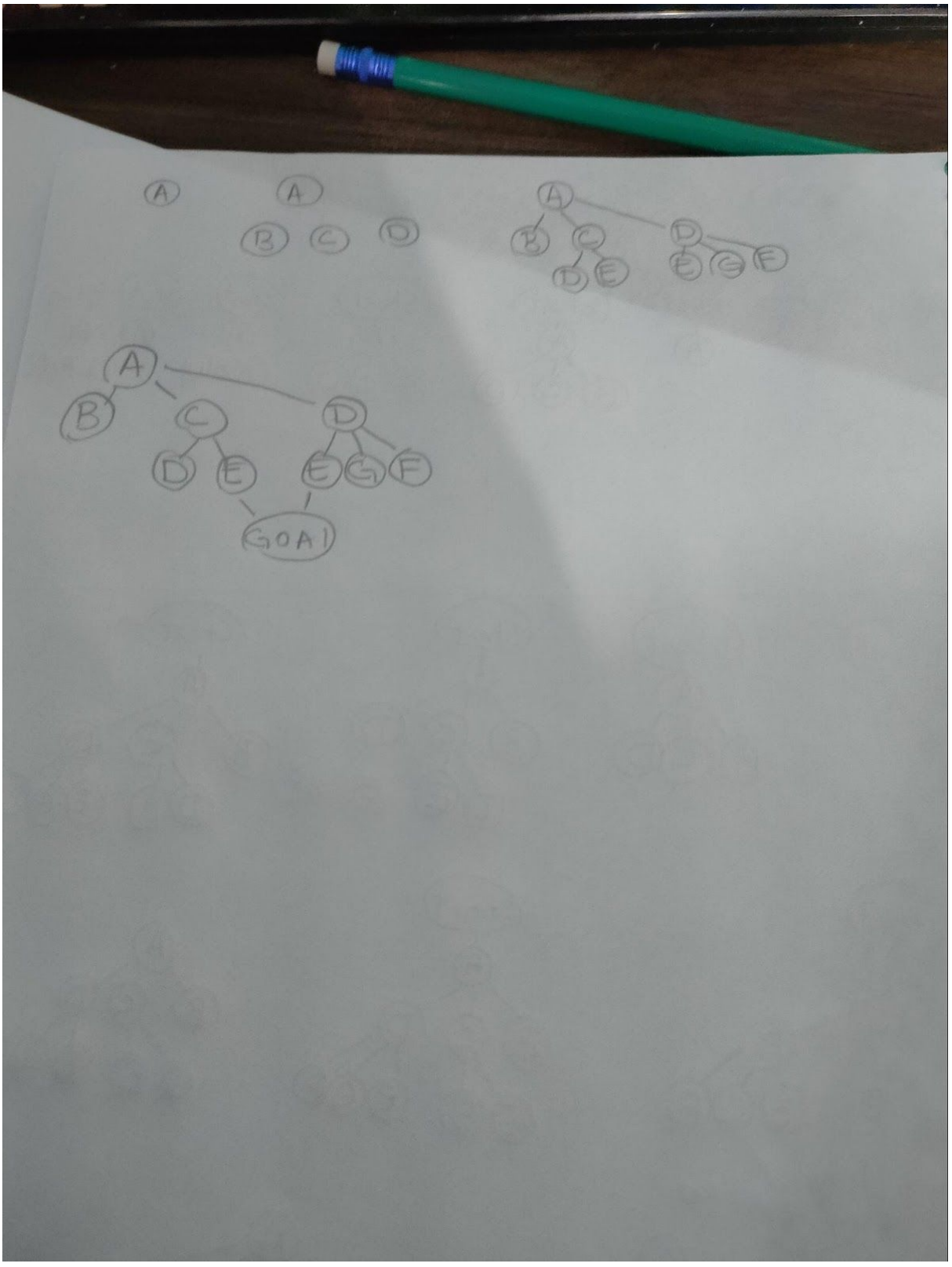
F: E F G D, C, D, A





F: GOAL, E, F, G, D, E, C
D, A


start
A
B
C
D
E
F
G



Para mim a busca em largura deu o melhor resultado, pois considerei ordem alfabética de escolha do nodos porém se escolher por peso pode dar outro algoritmo, mas o de aprofundamento iterativo eu poderia encontrar o com menor custo porém mais nodos eu abro gerando custo de processamento se eu considerar o limite como alto.

Aulas 09 e 10 (a resposta para o exercício 11 – a ser proposto na aula 10 - será usada como critério de "presença" nas aulas 09 e 10 e determinação do conceito final na disciplina)

(11) Trabalho de implementação do algoritmo A* (a ser definido)
