Código de honra e conduta discente:
Eu, Lucas Mateus Fernandes , matrícula 0035411, prometo pela minha honra que fui honesto e não trapaceei nessa avaliação passando ou recebendo cola.

Formiga, MG, 11 de março de 2021.

Exercício 1)

```
Função maiores3elementosStart (k):
        inicio = 0
        fim = tamanho(k) // tamanho dos elementos de K
        resposta = maiores3elementos (k, inicio, fim)


Função maiores3elementos (k, inicio, fim):

        Se inicio < fim entao
                meio = (inicio + fim) /2
                res1 = maiores3elementos (k, inicio, meio)
                res2 = maiores3elementos (k, meio+1, fim)
                retorna resultado(res1,res2)

        Se não
                vetPossivel = [ k[inicio], -INF, -INF ]
                para cada  i em (inicio até fim) :
                        para cada  j  em (0 até 3):
                                Se k[i] > vetPossivel[0] então
                                        vetPossivel[0] = k[i]
                                Se não
                                        Se k[i] > vetPossivel[1] então
                                                vetPossivel[1] = k[i]
                                        Se não
                                                Se k[i] >  vetPossivel[2] então
                                                        vetPossivel[2] = k[i]
                retorna  vetPossivel

Função resultado(res1, res2 ):

        resposta = [ res1[0], res1[1], res1[2] ]
        Troca = True
        Enquanto Troca == True
                Troca = False
                indice = 0
                Para cada i em res2:
                        Se i > resposta[indice]
                                resposta[indice] = i
                                i = i+1
                                Troca = True
                indice = 0
                Para cada i em res1:
                        Se i > resposta[indice]
```

```
                                resposta[indice] = i
                                i = i+1
                                Troca = True

        retorna resposta
```

questão 2)

Valores possíveis para ultimo = [N, S, L, O, NL, NO, SL, SO]

```
função checkBombaStart(x,y):
    Se Aqui(x,y) então:
            Achou = True
        retorna(x,y)

    Se Norte(x,y) então:
        Se Leste(x,y) então
            retorna checkBomba(x+1, y+1, 'NL')
        Se Oeste(x,y) então
            retorna checkBomba(x-1, y+1, 'NO')
        retorna checkBomba(x, y+1 ,'N')

    Se Sul(x,y) então:
        Se Leste(x,y) então
            retorna checkBomba(x+1, y-1, 'SL')
        Se Oeste(x,y) então
            retorna checkBomba(x-1, y-1, 'SO')
        retorna checkBomba(x, y-1, 'S')

    Se Leste(x,y) então
        retorna checkBomba(x+1, y, 'L' )

    Se Oeste(x,y) então
        retorna checkBomba(x-1, y, 'O')


função checkBomba(x,y,ultimo):
    Se Aqui(x,y) então:
            Achou = True
        retorna(x,y)

    Se Ultimo == 'N' então
        retorna checkBomba(x, y+1, 'N')

    Se Ultimo == 'NL' então
        Se Norte(x,y) então:
            Se Leste(x,y) então
                retorna checkBomba(x+1, y+1, 'NL')
            retorna checkBomba(x, y+1, 'N')
        retorna checkBomba(x+1, y, 'L')
```

```
Se Ultimo == 'NO' então
    Se Norte(x,y) então:
        Se Oeste(x,y) então
            retorna checkBomba(x-1, y+1, 'NO')
        retorna checkBomba(x, y+1, 'N')
    retorna checkBomba(x-1, y, 'O')

Se Ultimo == 'S' então
    retorna checkBomba(x, y-1, 'S')

Se Ultimo == 'SL' então
    Se Sul(x,y) então:
        Se Leste(x,y) então
            retorna checkBomba(x+1, y-1, 'SL')
        retorna checkBomba(x, y-1, 'S')
    retorna checkBomba(x+1, y, 'L')

Se Ultimo == 'SO' então
    Se Sul(x,y) então:
        Se Oeste(x,y) então
            retorna checkBomba(x-1, y-1, 'SO')
        retorna checkBomba(x, y-1, 'S')
    retorna checkBomba(x-1, y, 'O')

Se Ultimo == 'L' então
    retorna checkBomba(x+1, y, 'L')

Se Ultimo == 'O' então
    retorna checkBomba(x-1, y, 'O')
```

Questão 3)
Tem ‘N’ delegados sendo que maioria majoritária é de um único partido ou seja a quantidade de partidos vai de 1 até [(n/2) -1]

N = Quantidade de delegados
check = Função que verifica se dois delegados são ou não do mesmo partido
vetCheck = [1 … N]// Candidatos que não se sabem quais os partidos
vetPartidos = [[]] // Vetor de vetor porem inicializa Vazio
maiorPartido = vetPartidos[0] // Indica qual o partido com mais delegados

```
para cada i em vetCheck:
        vetPartido = [i] // i é o primeiro candidato do partido
        para cada J em vetCheck:
                Se i != J então // para não verificar um candidato com ele Mesmo
                        result = check(i,J) // Verifica se i e j são ou não do mesmo partido
                        Se result então
                                remove j de vetCheck
                                vetPartido =  vetPartido + J // Inclui J como candidato do
partido
                        Se tamanho(vetPartido) > tamanho(maiorPartido) então
                                maiorPartido = vetPartido
```

O partido com maior quantidade é o partido presente em maiorPartido

Questão 4)a)

Dado um conjunto problema 'p' passar os elementos para um vetor 'vetor' contendo o peso
vetor = Ordenar(vetor)
tamanho = Tamanho(vetor)

Se umLado(vetor, tamanho) == Verdadeiro então
    retorne Verdadeiro
Se não:
    retorne Falso

Função subConjuntoValido(vet, tamanho, total):
    Se total == 0 então
        retorna Verdadeiro
    Se tamanho == 0 e total != 0 então
        retorna Falso
    Se vetor[tamanho-1] > total então
        retorna subConjuntoValido(vetor, tamanho-1, total)
    retorna subConjuntoValido(vetor, tamanho-1, total) ou subConjuntoValido(vetor, tamanho-1, total - vetor[tamanho-1])

Função umLado(vet, tamanho):
    total = somatório(vet)
    Se soma for par então
        retorna Falso
    retorna subConjuntoValido(vetor, tamanho, total/ 2)

b)
Dado um conjunto problema 'p' passar os elementos para um vetor 'vetor' contendo o peso
vetor = Ordenar(vetor)
tamanho = Tamanho(vetor)
Se umLado(vetor, tamanho) == Verdadeiro:
        retorna Verdadeiro
Se não:
        retorna Falso
função findPartition(vetor, tamanho):
    soma = Somatorio(vet)
    Se soma não for par
        retorna Falso
    part = Matrix de dimensão [tamanho+1] por [(soma/2)+1] com valores iniciais de Verdadeiro
    Para cada i em (1 até soma//2 + 1):
       part[i][0] = False
    Para cada i em (1 até (soma/2 + 1))então
       Para cada j em (1 até tamanho+1)então
          part[i][j] = part[i][j-1]
             Se i >= vetor[j -1] então
                part[i][j] = (part[i][j] ou part[i- vetor[j – 1]][j-1])
       return part[soma / 2][n]

c)
Dado um conjunto problema 'p' passar os elementos para um vetor 'vetor' contendo o peso
vetor = OrdenarDescrescente(vetor)
tamanho = Tamanho(vetor)
soma = Somatório(vetor)
resposta = [0]
Para cada i em (vetor) então
        Se (i + Somatório(resposta)) <= (soma/2) então
          Adiciona i ao vetor respostas
Retorna (Somatório(resposta) == soma/2)

Sim resolve pois sempre parte de um estado subotimo que é onde o maior elemento possível foi adicionado em um dos lados e se este lado estiver balanceado no final significa que o outro lado também estara

Questão 5)

```
função distancia(cidadeOrigem,cidadeDestino)
    Dado o grafo g retorna a distancia entre cidadeOrigem,cidadeDestino assumindo que
existe uma aresta de Origem para destino

função indexCidade(vetorCidade, cidadeAtual)
    //Verifica qual o indice da cidade Atual dentro do vetor Cidades
    index = 0
    Para cada cidade em  vetorCidade
        Se cidadeAtual == cidade então
            Sai do laço de repetição
        Se não
            indexCidade = indexCidade + 1
    retorna index

função maxCidade(vetorCidade, cidadeAtual, tanque)

    indexInicial= indexCidade( vetorCidade, cidadeAtual)
    indexFinal =   indexInicial
    index =  indexInicial

    Para cada cidade em ( vetorCidade[ indexInicial] até
vetorCidade[ Tamanho(vetorCidade)]  )
        Se cidade == vetorCidade[ Tamanho(vetorCidade)] então
            retorna (index,indexFinal)
        Se tanque - distancia(cidade, vetorCidade[ index+1]) >= 0 então
          tanque = tanque – distancia(cidade, vetorCidade[ index+1])
              index = index + 1
              indexFinal = index
        Se não
              return (index,indexFinal)


função  ondeAbastecer(range, vetorPrecoGasolina)
    indexAtual = range[0]
    indexMenorValor =  indexAtual
    menorValor = vetorPrecoGasolina[ range[0] ]
    Para cada valor em (vetorPrecoGasolina[ range[0] ] até vetorPrecoGasolina[ range[1] ] )
        Se   valor < menorValor então
          indexMenorValor =   indexAtual
          menorValor = vetorPrecoGasolina[ indexMenorValor ]
         indexAtual =  indexAtual + 1
    retorna  indexMenorValor




função Resolve(g,k,r,p)
    totalCidades = Tamanho(r)
    capacidade = k
    tanque = k
```

```
vetorCidade = r
vetorPrecoGasolina = p
abastecimento = []


Para cada cidade em ordenCidades:
   index = indexCidade(vetorCidade, cidade)

   //Caso não esteja na ultima cidade
   Se index != Tamanho( vetorCidade ) - 1 então

      // Verifico se consigo ir para a próxima cidade
      Se distancia(cidade, vetorCidade[index+1] ) > tanque
        retorna Falso

      //verifico até onde é possível chegar com determinada gasolina
      range = maxCidade(vetorCidade, cidade, tanque)

      //Onde há a gasolina mais barata
      indexCidadeAbastecimento = ondeAbastecer(range,vetorPrecoGasolina)

      //Abastece caso necessário
      Se index == indexCidadeAbastecimento e tanque < capacidade então
          adiciona a lista de abastecimento = (index, capacidade - tanque)
          tanque = capacidade
      Se não
          adiciona a lista de abastecimento = (index, 0)

     //Movimenta até a próxima cidade
     tanque = tanque - distancia(cidade, vetorCidade[index+1] )

   //Chegou ao final
   Se não
      // Houve abastecimento em excesso
      Se tanque != 0
       // Procura onde foi o ultimo abastecimento
       Para cada ponto em ( OrdemInversa(abastecimento) )
            Se ponto[1] != 0
               //Remove o excesso abastecido
               ponto[1] = ponto[1] – tanque

  //É factivel e retorna uma lista com os pontos onde se deve abastecer e quanto se deve
abastecer
  retorna abastecimento
```