Nomes: Ryan Eduardo Mansur Vasconcelos e Thiago de Assis Lima

Questão 1)

Abaixo imagem da parte do código onde se implementa a programação dinâmica para resolver o problema.

```
int x1, e1;
int x2, e2;

e1 = linha1.get(0).getValor(); //valor de entrada linha 1
e2 = linha2.get(0).getValor(); //valor de entrada linha 2
x1 = linha1.get(tam+1).getValor(); //valor de saida linha 1
x2 = linha2.get(tam+1).getValor(); //valor de saida linha 2

//algoritmo mostrado na aula
f1[1] = e1 + linha1.get(1).getValor();
f2[1] = e2 + linha2.get(1).getValor();
for(int j=2;j<=tam; j++){
    if(f1[j-1] + linha1.get(j).getValor() <= f2[j-1] + linha2.get(j-1).getTransporte() + linha1.get(j).getValor()){
        f1[j] = f1[j-1] + linha1.get(j).getValor();
        l1[j] = 1;
    }
    else{
        f1[j] = f2[j-1] + linha2.get(j-1).getTransporte() + linha1.get(j).getValor();
        l1[j] = 2;
    }
    if(f2[j-1] + linha2.get(j).getValor() <= f1[j-1] + linha1.get(j-1).getTransporte() + linha2.get(j).getValor()){
        f2[j] = f2[j-1] + linha2.get(j).getValor();
        l2[j] = 2;
    }
    else{
        f2[j] = f1[j-1] + linha1.get(j-1).getTransporte() + linha2.get(j).getValor();
        l2[j] = 1;
    }
}
if(f1[tam] + x1 <= f2[tam] + x2){
    f = f1[tam] + x1;
    l = 1;
}
else{
    f = f2[tam] + x2;
    l = 2;
}
```

A grande característica da programação dinâmica é a ideia de guardar todas as sub soluções.
No caso do exemplo, o algoritmo só visita cada estação uma vez, calcula o caminho até ela e guarda a solução para as próximas estações. E apenas no final, ao comparar as soluções diz qual é a solução do problema.
Assim, o programa gasta bem menos tempo do que, por exemplo, um programa recursivo onde as soluções seriam recalculadas a cada chamada para outra estação.

Abaixo imagem de parte do código onde se implementa o algoritmo de dijkstra, uma resolução gulosa para o problema.

```java
int[] distanciasMenores = new int[numVertices]; //vetor para registrar os menores pesos
boolean[] adicionados = new boolean[numVertices]; //vetor para marcar os vertices já marcados
for (int vertIndice = 0; vertIndice < numVertices; vertIndice++){
    distanciasMenores[vertIndice] = Integer.MAX_VALUE;
    adicionados[vertIndice] = false;
}
distanciasMenores[inicio] = 0;
int[] antecessores = new int[numVertices];
antecessores[inicio] = -1;
for (int i = 1; i < numVertices; i++){
    int vertice_proximo = -1;
    int menor_peso = Integer.MAX_VALUE; //inicializa o menor peso como infinito
    for (int vertIndice = 0; vertIndice < numVertices; vertIndice++) {
        //checa se vertice já foi adicionado, se nao, entao compara com o menor peso registrado
        if (!adicionados[vertIndice] && distanciasMenores[vertIndice] < menor_peso) {
            vertice_proximo = vertIndice;
            menor_peso = distanciasMenores[vertIndice];
        }
    }
    if(vertice_proximo != -1){
        adicionados[vertice_proximo] = true;

        for (int vertIndice = 0; vertIndice < numVertices; vertIndice++) {
            int peso = mat[vertice_proximo][vertIndice];

            if (peso > 0 && ((menor_peso + peso) < distanciasMenores[vertIndice])) {
                //atualiza os vetores de antecessor e distancia com os novos valores corretos
                antecessores[vertIndice] = vertice_proximo;
                distanciasMenores[vertIndice] = menor_peso + peso;
            }
        }
    }
}

this.menor_peso = distanciasMenores[fim];
this.distanciasMenores = distanciasMenores;
```

O algoritmo guloso, busca a solução ótima em cada pesquisa, e após isso verifica se a solução gerada é uma solução ótima geral.
Assim funciona o algoritmo de dijkstra, que pesquisa em cada vértice, o próximo vértice onde o caminho é menos custoso.
Ainda assim, para implementar o algoritmo no problema das linhas de montagem, foi necessário chamar o algoritmo 4 vezes, analisando o menor caminho de e1 a x1, e1 a x2, e2 a x1 e e2 a x2, e a partir disso comparar os resultados, sendo o melhor deles o resultado do problema.

Questão 2)

Os resultados obtidos para ambos métodos foram os mesmos. O resultado do algoritmo guloso foi mapeado considerando um vetor, que começava em e1 e ia até x2. Abaixo imagens da saída dos programa usando os dois métodos.

```
Primeiro caso - PD>>>

    j  1 2 3 4 5 6
f1[j]: 9 18 20 24 32 35
f2[j]: 12 16 22 25 30 37
O valor do caminho minimo e:
f* = 38
    j  1 2 3 4 5 6
l1[j]: 1 2 1 1 2
l2[j]: 1 2 1 2 2
A linha inicial e:
l* = 1


Dijkstra>>>
Menor peso e1 -> x1: 38
Caminho:  -> 0 -> 1 -> 2 -> 12 -> 4 -> 14 -> 15 -> 7 -> 8
Menor peso e1 -> x2: 39
Caminho:  -> 0 -> 1 -> 2 -> 12 -> 4 -> 14 -> 15 -> 16 -> 17
Menor peso e2 -> x1: 39
Caminho:  -> 9 -> 10 -> 11 -> 12 -> 4 -> 14 -> 15 -> 7 -> 8
Menor peso e2 -> x2: 40
Caminho:  -> 9 -> 10 -> 11 -> 12 -> 4 -> 14 -> 15 -> 16 -> 17
```

Para o caso exemplo, obtivemos a solução esperada, o melhor caminho com custo de 38. No guloso, o melhor caminho é o primeiro e1 a x1.

```
Segundo caso - PD>>>

    j  1 2 3 4 5 6
f1[j]: 8 15 24 29 38 49 55 57 59
f2[j]: 8 11 20 31 35 44 47 59 62
O valor do caminho minimo e:
f* = 65
    j  1 2 3 4 5 6
l1[j]: 1 2 1 1 1 2 2 1
l2[j]: 2 2 2 2 2 2 2 1
A linha inicial e:
l* = 1


Dijkstra>>>
Menor peso e1 -> x1: 66
Caminho:  -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 18 -> 19 -> 20 -> 9 -> 10 -> 11
Menor peso e1 -> x2: 68
Caminho:  -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 18 -> 19 -> 20 -> 9 -> 22 -> 23
Menor peso e2 -> x1: 65
Caminho:  -> 12 -> 13 -> 14 -> 15 -> 4 -> 5 -> 18 -> 19 -> 20 -> 9 -> 10 -> 11
Menor peso e2 -> x2: 67
Caminho:  -> 12 -> 13 -> 14 -> 15 -> 4 -> 5 -> 18 -> 19 -> 20 -> 9 -> 22 -> 23
```

Para o exercício 1, obtivemos o caminho com custo de 65, sendo o guloso com o mesmo resultado, com caminho de e2 a x1.

```
Terceiro caso - PD>>>

    j  1 2 3 4 5 6
f1[j]: 15 21 19 27 32 35 42 54
f2[j]: 10 15 18 25 31 35 44 54
O valor do caminho minimo e:
f* = 62
    j  1 2 3 4 5 6
l1[j]: 1 2 1 1 1 1 1
l2[j]: 2 2 2 2 2 2 2
A linha inicial e:
l* = 1


Dijkstra>>>
Menor peso e1 -> x1: 67
Caminho:  -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10
Menor peso e1 -> x2: 68
Caminho:  -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 20 -> 21
Menor peso e2 -> x1: 62
Caminho:  -> 11 -> 12 -> 13 -> 14 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10
Menor peso e2 -> x2: 63
Caminho:  -> 11 -> 12 -> 13 -> 14 -> 4 -> 5 -> 6 -> 7 -> 8 -> 20 -> 21
```

Para o exercício 2, obtivemos o caminho com custo de 62, sendo o guloso com o mesmo resultado, com caminho de e2 a x1.