Centro Federal de Educação Tecnológica de Minas Gerais - Departamento de Computação
Laboratório de Algoritmos e Estruturas de Dados II
Prof. Thiago de Souza Rodrigues

Guilherme Moreira de Carvalho
Marcus Vinícius Diniz Ribeiro
20/08/2021

Relatório do Trabalho Prático 02

Programação Dinâmica

A solução implementada por meio da programação dinâmica otimiza o problema das linhas de montagem, uma vez que utiliza recursão com memoização, resolvendo cada sub-problema - sendo, nesse caso, o cálculo do tempo para completar cada etapa da montagem - e armazena o resultado para que possa ser usado para resolver o subsequente. Assim, ao fim do processamento, tem-se o tempo até a saída de ambas as linhas e o caso ótimo - menor tempo.

```java
// otimiza o processo de montagem considerando os tempos de cada estação e de
// transferência entre linhas
public int[] montagemOtima (int A1[], int[] A2, int[] T1, int[] T2) {
    // f*: caminhos mais rápidos até cada estação, começando, respectivamente,
    // pela linha 1 e pela linha 2
    // l*: linha pela qual se chega a cada estação mais rapidamente
    int[] f1 = new int[NUM_ESTACAO + 1]; int[] l1 = new int[NUM_ESTACAO + 1];
    int[] f2 = new int[NUM_ESTACAO + 1]; int[] l2 = new int[NUM_ESTACAO + 1];

    // e: tempo de entrada; x: tempo de saída
    int[] e = {A1[0], A2[0]};
    int[] x = {A1[NUM_ESTACAO + 1], A2[NUM_ESTACAO + 1]};

    // entradas
    f1[0] = e[0] + A1[1]; l1[0] = 1; l1[NUM_ESTACAO] = 1;
    f2[0] = e[1] + A2[1]; l2[0] = 2; l2[NUM_ESTACAO] = 2;

    // verifica, recursivamente, qual o caminho mais rápido a cada estação
    // > pela estação anterior da mesma linha ou
    // > pela estação anterior de outra linha acrescido o tempo de transporte
    for (int i = 1; i < NUM_ESTACAO; i++) {
        if (f1[i-1] + A1[i+1] <= f2[i-1] + T2[i-1] + A1[i+1]) {
            f1[i] = f1[i-1] + A1[i+1]; l1[i] = 1;
        } else {
            f1[i] = f2[i-1] + T2[i-1] + A1[i+1]; l1[i] = 2;
        }
        if (f2[i-1] + A2[i+1] <= f1[i-1] + T1[i-1] + A2[i+1]) {
            f2[i] = f2[i-1] + A2[i+1]; l2[i] = 2;
        } else {
            f2[i] = f1[i-1] + T1[i-1] + A2[i+1]; l2[i] = 1;
        }
    }

    // saídas
    f1[NUM_ESTACAO] = f1[NUM_ESTACAO - 1] + x[0];
    f2[NUM_ESTACAO] = f2[NUM_ESTACAO - 1] + x[1];

    // o tempo gasto é o mínimo entre as saídas
    this.T_OUT = min(f1[NUM_ESTACAO], f2[NUM_ESTACAO]);

    // retorna o percurso com menor tempo
    if (f1[NUM_ESTACAO] <= f2[NUM_ESTACAO]) {
        return l1;
    } else {
        return l2;
    }
}
```

Algoritmo Guloso

O algoritmo guloso utiliza a iteração e apenas considera o resultado atual, perdendo os parâmetros para comparação entre os tempos de processos anteriores.

```
// estima o tempo gasto no processo de montagem considerando os tempos de cada estação e de
// transferência entre linhas
public int[] montagemOtima (int A1[], int[] A2, int[] T1, int[] T2) {
    // estação pela qual se chega a cada estação pelo caminho mais rápido
    int[] l = new int[NUM_ESTACAO + 2];

    int[] e = {A1[0], A2[0]};
    int[] x = {A1[NUM_ESTACAO + 1], A2[NUM_ESTACAO + 1]};

    // tempo de entrada mais rápido
    if (e[0] + A1[1] <= e[1] + A2[1]) {
        l[0] = 1; l[1] = 1;
        T_OUT += e[0] + A1[1];
    } else {
        l[0] = 2; l[1] = 2;
        T_OUT += e[1] + A2[1];
    }

    // verifica, iterativamente, qual o caminho mais rápido a se seguir
    // > continua na linha atual ou
    // > transporta para outra
    for (int i = 2; i <= NUM_ESTACAO; i ++) {
        if (l[i-1] == 1) {
            if (T_OUT + A1[i] <= T_OUT + A2[i] + T1[i-2]) {
                l[i] = 1;
                T_OUT += A1[i];
            } else {
                l[i] = 2;
                T_OUT += A2[i] + T1[i-2];
            }
        } else {
            if (T_OUT + A2[i] <= T_OUT + A1[i] + T2[i-2]) {
                l[i] = 2;
                T_OUT += A2[i];
            } else {
                l[i] = 1;
                T_OUT += A1[i] + T2[i-2];
            }
        }
    }

    // tempo de saída mais rápido
    if (l[NUM_ESTACAO] == 1) {
        l[NUM_ESTACAO + 1] = 1;
        T_OUT += x[0];
    } else {
        l[NUM_ESTACAO + 1] = 2;
        T_OUT += x[1];
    }

    // retorna o percurso
    return l;
}
```

Resultados

Para os dois exemplos, a programação dinâmica retornou os percursos de menor tempo - como era esperado.