Mestrado Profissional em Matemática, Estatística e Computação Aplicado à Industria (MECAI)

Trabalho de MAI5001 - Introdução a Ciências da Computação

Analise e Desempenho em programação Dinâmica

Prof. Dr.: Adenilso Simões

Aluno: João Carlos Batista
N°USP: 6792197

USP – São Carlos 18/07/2017

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1: Descrição da técnica

1.1. Descrição do algoritmo

A programação dinâmica é uma técnica de otimização aplicada em problemas que podem ser divididos em subproblemas menores em comum, o objetivo é evitando o recalculo. A técnica consiste em guardar os resultados de determinados cálculos numa tabela ou memória para que haja o reaproveitamento desse resultado, por esse motivo é mais eficiente usar em casos onde ocorre a decomposições de subproblemas que compartilham os mesmos subproblemas. A técnica resolve problema maior em menor tempo de execução do algoritmo e não usar muito recurso de memória. A figura 1 representa a resolução de um problema inicial que é subdividido em subproblemas menores, alguns desses subproblemas se repetem, ocorrendo assim *overlap*.

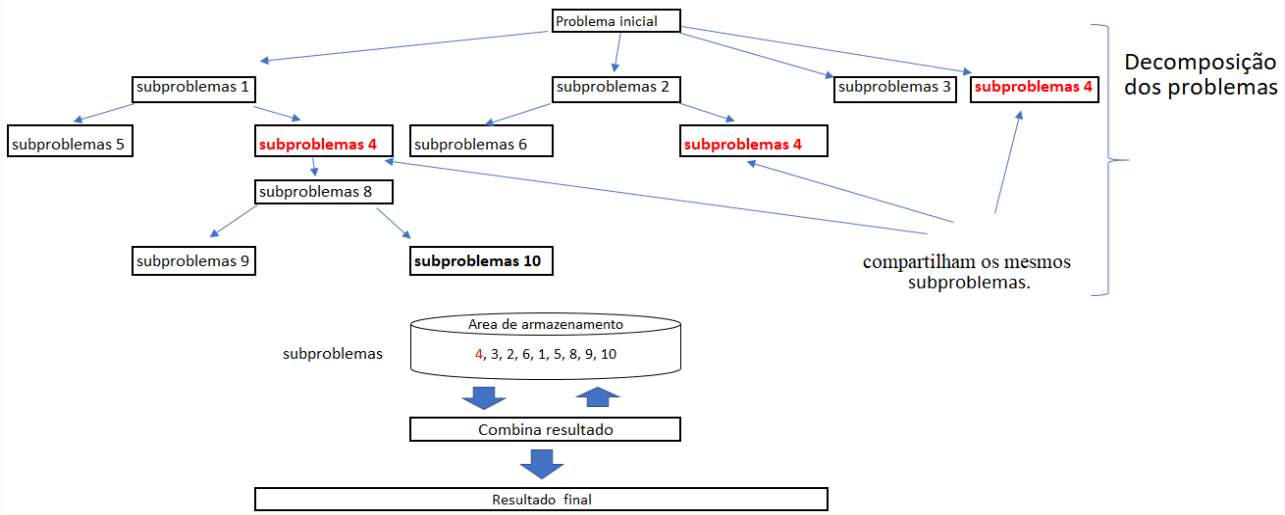


Figura 1:Dinâmica do algoritmo.

1.1.1 Características

- A cada passo são guardados subsoluções não repetidas que certamente faria parte da solução ótima do problema.
- Reduz drasticamente o número total de subproblemas a serem resolvidos com o uso da técnica de memorização seja com uso de tabelas ou memorias.

2. Problema da mochila

2.1 Codificação em C#

```
static public void combinar_item_na_muchila(int contar_tem, int Num_elementos, int
total_de_combinacoes, int[,] capacidade, int[,] codigo_gray, int[] peso, int[]
preco)
{
    for (int j = 0; j < Num_elementos; j++)
    {//  faz toda a combinacoes de item  (muchila  0/1 )
        capacidade[contar_tem, 0] = capacidade[contar_tem, 0] +
codigo_gray[contar_tem, j] * peso[j];
        capacidade[contar_tem, 2] = capacidade[contar_tem, 2] +
codigo_gray[contar_tem, j] * preco[j];
        capacidade[contar_tem, 1] = contar_tem;
    }
    if (contar_tem < total_de_combinacoes)
    {   contar_tem = contar_tem + 1;
        combinar_item_na_muchila(contar_tem, Num_elementos, total_de_combinacoes,
capacidade, codigo_gray, peso, preco);
    }
}

/// ------------------------------------------------------------------------

static public void combinar_item_na_muchila_dinamicamente(int total_de_linhas,
int[,] codigo_gray, int[,] capacidade, int[] peso, int[] preco, int colunas)
{ int contador = 0, linhas = 0, linha_j = 0;
  for (int i = 1; i <= total_de_linhas; i++)///  verifica a  tabela  verdade
  {
      contador = contador + 1;
      if (contador >= total_de_linhas / 2 - 1)
      {
          if (linhas < total_de_linhas / 2)
          {   linhas = linhas + 1;        }
      }
      for (int j = 0; j < colunas; j++)        ///  verifica a  tabela  verdade
      {
         if (contador < total_de_linhas / 2)    ///capacidade3[i,0]
         { capacidade[i, 0] = capacidade[i, 0] + codigo_gray[i, j] * peso[j];
           capacidade[i, 2] = capacidade[i, 2] + codigo_gray[i, j]*preco[j];
           capacidade[i, 1] = i;
         }
         else
         { capacidade[contador, 0] = codigo_gray[i, j] * peso[j] + capacidade[i, 0];
           capacidade[i, 2] = capacidade[i, 2] + codigo_gray[i, j]* preco[j];
           capacidade[i, 1] = i;
         }
      }
  }
}
```

2.2 Situação de interesse comercial

Uma aplicação onde há problema de otimização combinatória, a resolução do problema tem como objetivo a maximização do lucro, atender a capacidade de carga da mochila e levar o máximo do produto de maior valor.

2.3 Uso de código gray

O uso de código gray viabiliza obter sequências de combinações de item na mochila e também permite tomar decisões e reaproveitamento de resultado, reduzindo assim o número de estágio. Podemos verificar na figura 2 que existe repetições de coluna de zeros e uns

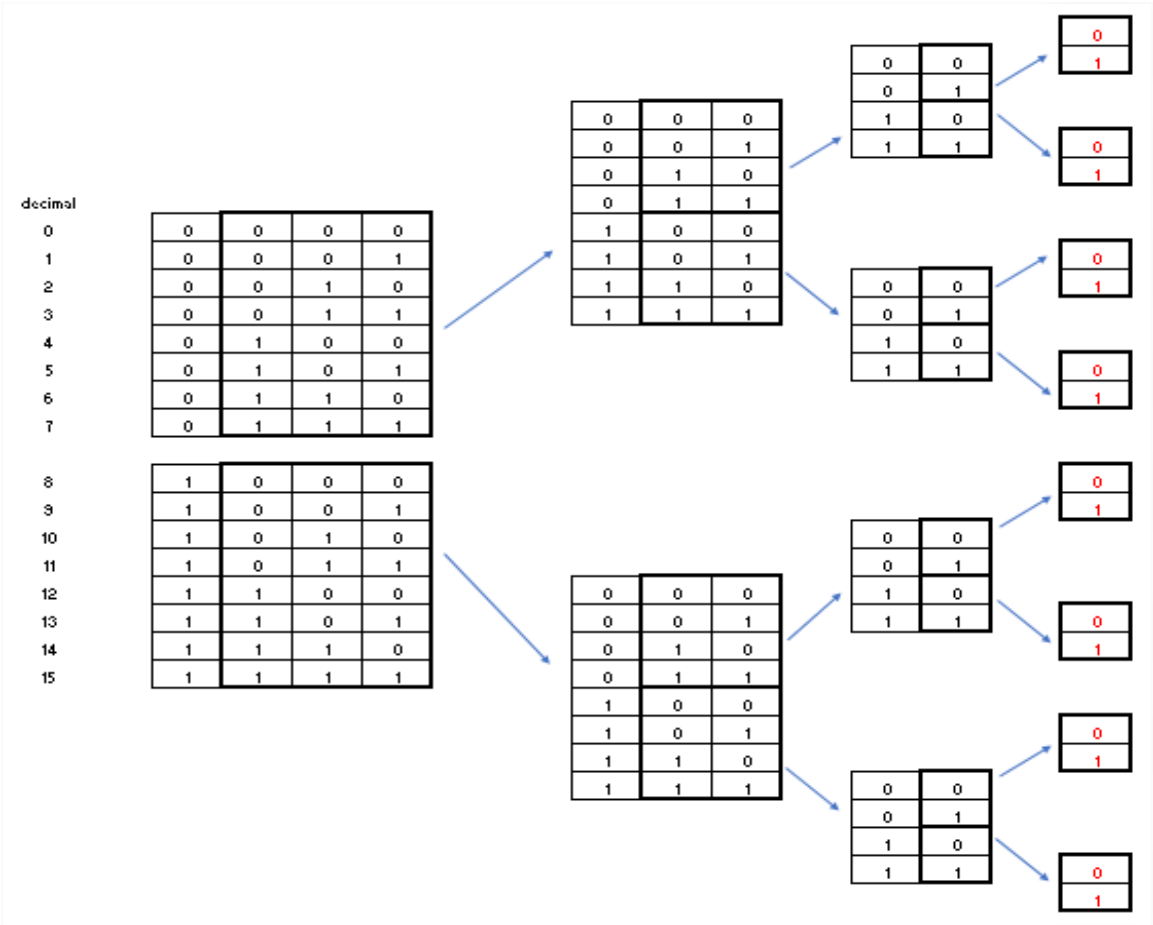


Figura 2: Código Gray

Parte do código onde há o uso de tabela (uso de recalculo) para fazer a combinação e soma de peso e preço está na função matriz de código gray.

2.4 Aplicação da mochila 0/1

	1	2	3	4	5	....
Item	A	B	C	D	E	
Peso(Kg)	7	2	15	5	11	
Preço(R$)	25	14	42	71	151	
Binário	0/1	0/1	0/1	0/1	0/1	

Tabela: Item da mochila.

A linha binária indica se o item vai na mochila e representado por 1, caso o item não vai na mochila e representado por 0.

Exemplo de cálculo:

Peso (2) = 1.A+0.B+1.C+ 1.D+1.E = 38 (capacidade de mochila = 40) Max R$ 289,00

Exemplo de combinação de todas as possibilidades e mostrado na figura 3:

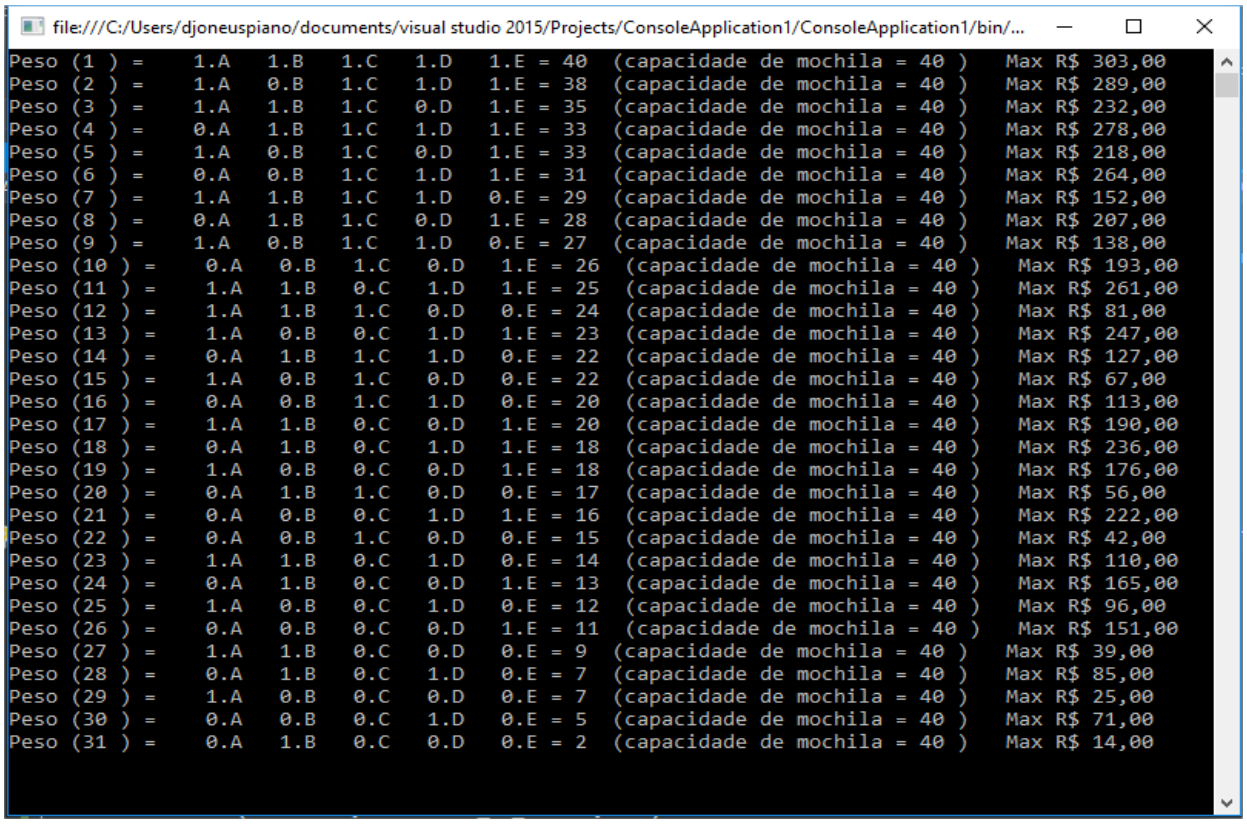


Figura 3: Mochila preenchida

E possível perceber se do dobrarmos a função Peso(18) teremos um Max de R$ 472,00, fazendo um uso de 36kg

Peso (18) = 0.A 1.B 0.C 1.D 1.E = 18 (capacidade de mochila = 40) Max R$ 236,00

3 Problema da linha de montagem

Neste problema, o objetivo é encontrar o caminho mais rápido passando por todas as estações, ou seja, cada estação tem um tempo de produção ou manutenção de um produto. Cada estação contém duas linhas de montagem onde são executadas as tarefas, um mais rápido e outras mais lenta. O algoritmo tem que traçar uma trajetória com a finalidade de reduzir o tempo de produção, podendo mudar de linha de produção, porem passando por todas as estações.

A técnica permite reconstruir a solução pouco a pouco e tomar decisões no decorrer do cominho para reduzir o tempo total, é possível redefinir trajetória no meio do caminho forçando-o passar por um determinado linha de montagem. O recalculo é ajustado automaticamente, sem a necessidade de refazer todos os cálculos anteriores.

3.1 Codificação em C# problema da linha de montagem

```
static public void linha_Producao_redefinida(int estacao, int linha, int
estacao_inicio, int[,] producao, int[,] percurso_antes, int[,] percurso)
        {  // linha de produca redefinica dinamicamente
            int total = percurso[4, estacao_inicio - 2]; //  pega  o  tempo anterior
            int linha_inicio = 5;   // seta a linha  onde contém  os dados  de
percursos
            percurso[linha_inicio, estacao_inicio] = 1;
            int frag = 0;


            for (int i = 0; i < linha - 1; i++)
            {
                for (int j = estacao_inicio - 1; j <= estacao; j++)      // entrada
de  dados
                {
                    int tempo = 1000000000;
                    int linha_de_montagem = 0;

                    for (int k = 0; k < linha; k++)      // entrada de  dados
                    {
```

```
                    if (producao[k, j] < tempo)
                    {
                        if (frag == 0)
                        {
                            if (0 == producao[0, j])
                            {
                                tempo = producao[0, j];
                                linha_de_montagem = k;

                                Console.Write("tempo 0 " + tempo + "\n");
                            }
                            if (1 == producao[1, j])
                            {
                                tempo = producao[1, j];
                                linha_de_montagem = k;
                                Console.Write("tempo 1 " + tempo + "\n");
                            }
                            frag = 1;
                        }
                        else
                        {
                            tempo = producao[k, j];
                            linha_de_montagem = k;

                        }
                    }
                }
                if (percurso[5, j] == 1 && j != 0)// voltar
                {
                    total = total + tempo;
                    percurso_antes[0, j] = percurso[0, j];
                    percurso_antes[1, j] = percurso[1, j];
                    percurso_antes[3, j] = percurso[3, j];
                    percurso_antes[4, j] = total;
                }
                else
                {
                    total = total + tempo;
                    percurso_antes[0, j] = linha_de_montagem;
                    percurso_antes[1, j] = tempo;
                    percurso_antes[3, j] = j;
                    percurso_antes[4, j] = total;


                }
            }
        }
    }
```

3.2 Aplicação da Linha de montagem

A figura 4 representa uma situação onde há 2 linhas de montagem e 6 estações.

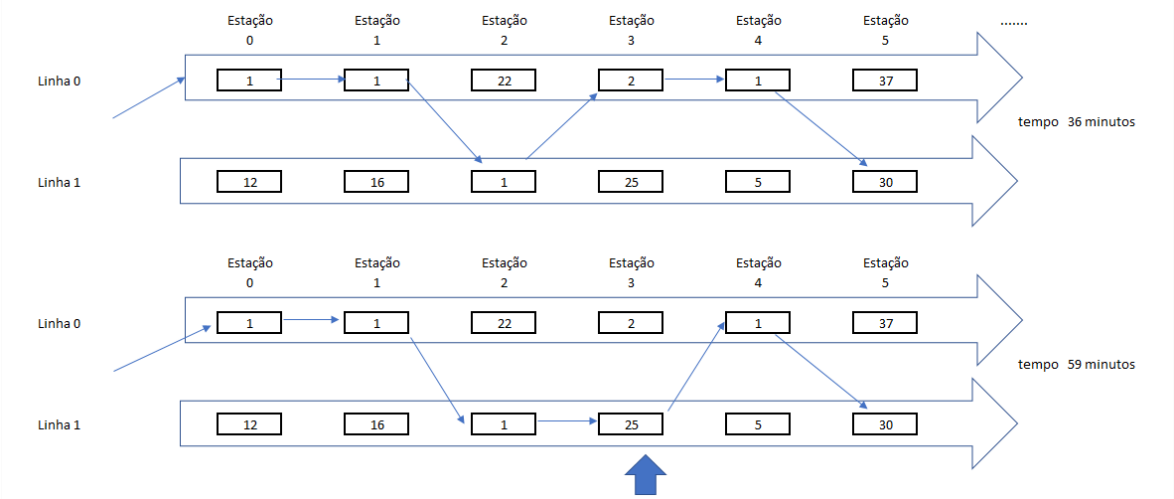


Figura 4: Linha de montagem

O primeiro percurso da linha de montagem foi traçado pelo algoritmo, a sequência de produção segue para a linha mais rápida o que resulta num tempo de 36 minutos. Já a segunda linha de produção foi redefinida, forçando-o o algoritmo passar pela linha 1 da estação 3 o qual resultou num tempo de 59 minutos como podemos verificar na figura 5 a seguir.

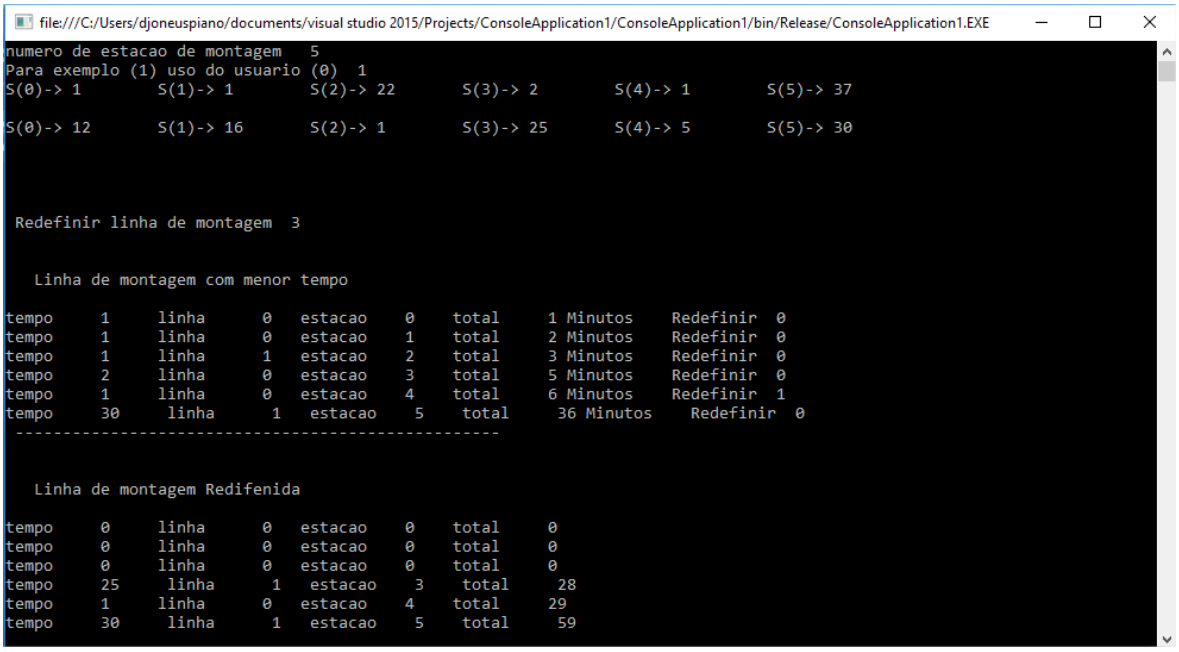


Figura 5: Linha de montagem

4 Sequência de Fibonacci

A sucessão de Fibonacci é uma sequência de números inteiro, que é iniciada para o cálculo F(0)=0 e F(1)=1 , cada termo subsequente corresponde a soma de dois anteriores. A sequência tem aplicação na análise financeiro, ciência da computação e na teoria dos jogos. A figura 5 representa a espiral de nautilus onde podemos perceber a sequência de Fibonacci.

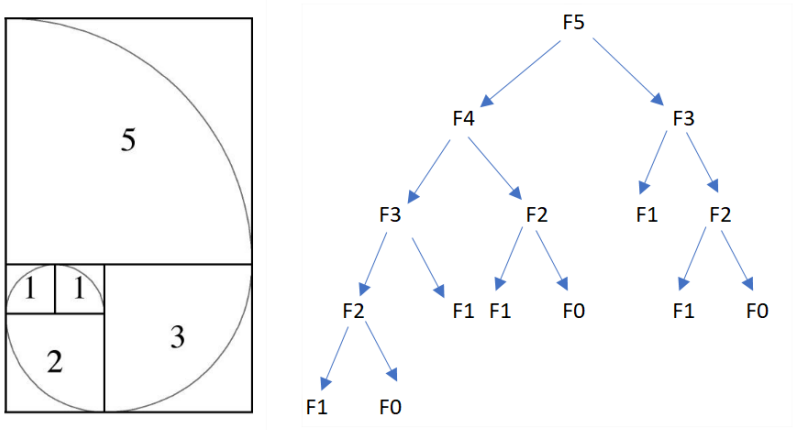


Figura 6:Espiral de nautilus.

4.1 Codificação em C# problema de Fibonacci

```
public static Int64 fibonacci_Recursivo(Int64 n)
{
  //  Console.Write("F1{" + n + "}  ");
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibonacci_Recursivo(n - 1) + fibonacci_Recursivo(n - 2);


}

public static Int64 fibonacci_Programacao_dinamica(Int64 n)
{
    Int64 number = n;
    Int64[] Fib = new Int64[number + 1];
        Fib[0] = 0;
        Fib[1] = 1;

        for (Int64 i = 2; i <= number; i++)
        {
            Fib[i] = Fib[i - 2] + Fib[i - 1];
          //  Console.Write("F2{" + i + "}  " + Fib[i]+" ");
        }

        return Fib[number];
```

```
}
public static Int64 Fibonacci(Int64 n)
{
    Int64 a = 0;
    Int64 b = 1;
        for (int i = 0; i < n; i++)
        {
        Int64 temp = a;
            a = b;
            b = temp + b;

           // Console.Write("F3{" + i + "}  " + b+ " ");
        }
        return a;
}
```

4.2 Calculo da sequência de fibonacci

Sequência de Fibonacci:

$$F(n) = F(n-1) + F(n-2)$$

$$F(5) = F(4) + F(3)$$

$$F(4) = F(3) + F(2)$$

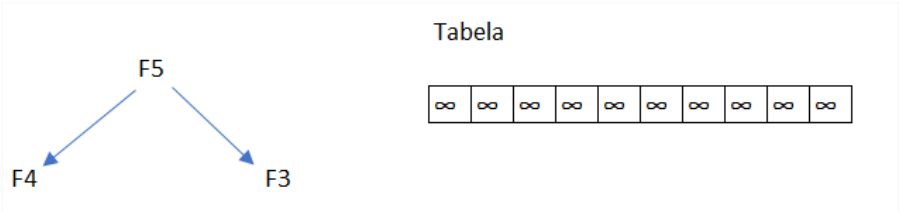
$$F(3) = F(2) + F(1)$$

$$F(2) = F(1) + F(0)$$

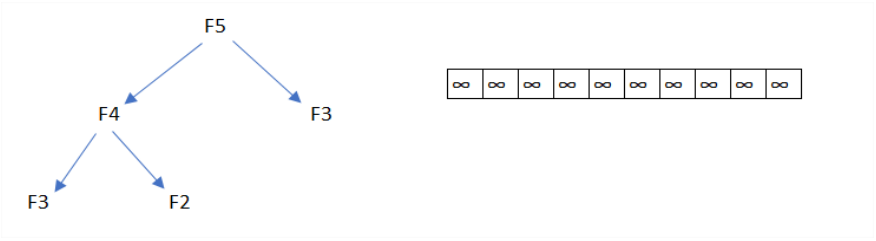
$$F(1) = 1$$

$$F(0) = 0$$

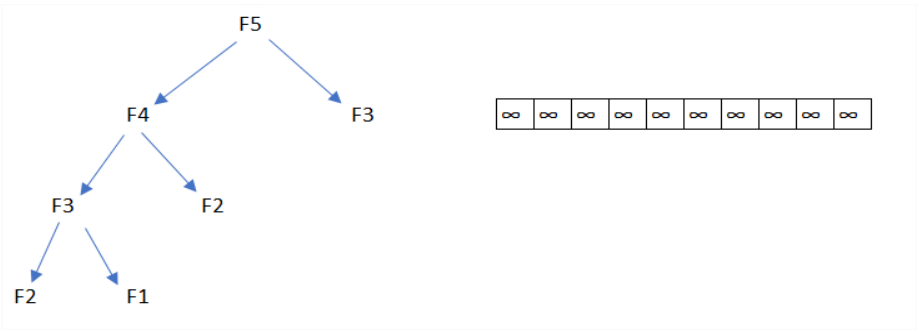
Execução da sequência usando programação dinâmica com uso de tabelas



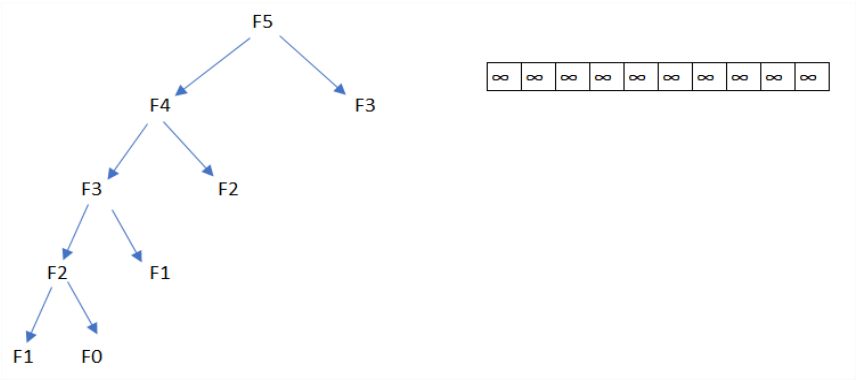
A iteração 1 expande para $F(5) = F(4) + F(3)$



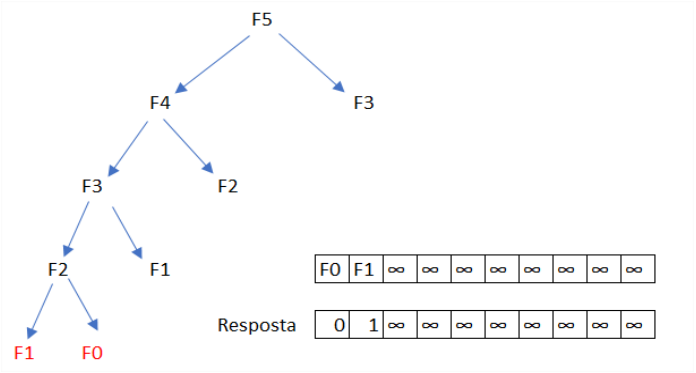
A iteração 2 expande para $F(4) = F(3) + F(2)$



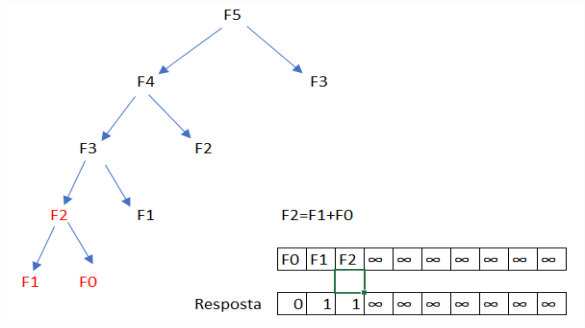
A iteração 3 expande para $F(3) = F(2) + F(1)$



A iteração 4 expande para $F(2) = F(1) + F(0)$

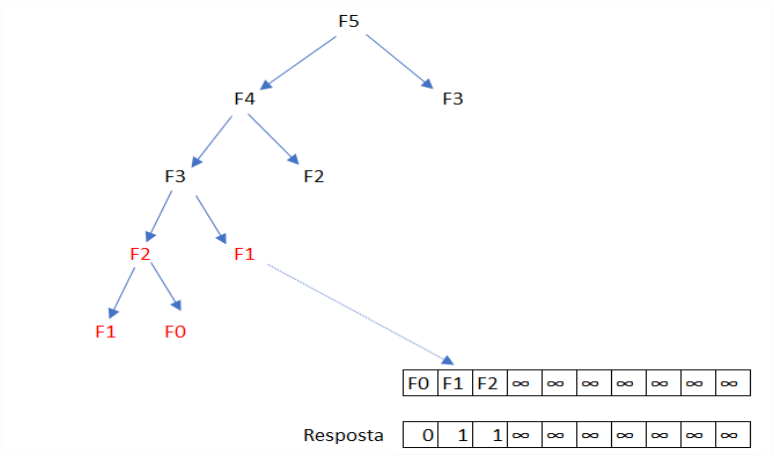


A iteração 5 já conhecemos $F(1) = 1$ *e* $F(0) = 0$ *agora podemos prosseguir para achar F(5) e sempre guardar o resultado na tabela*

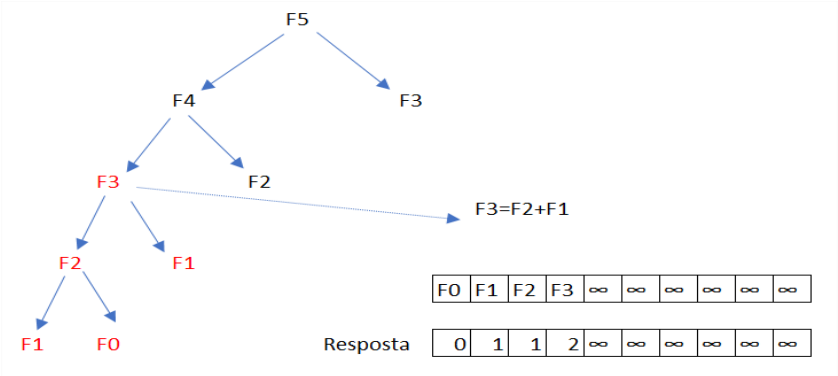


A iteração 6 podemos reaproveitar os cálculos de F0 e F1 para obtermos F2

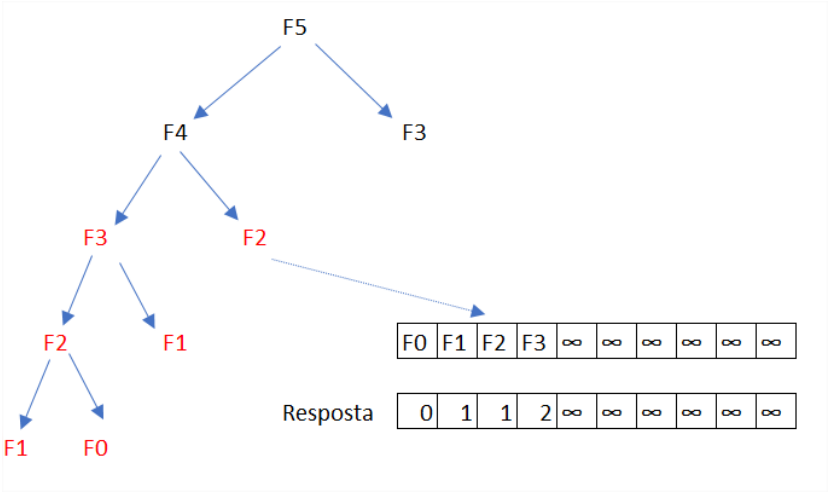
$F(2) = 1 + 0 = 1$



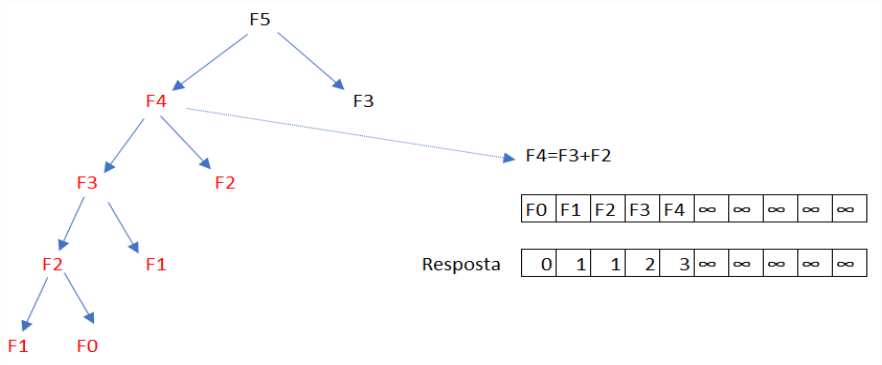
A iteração 7 verificamos na tabela que já tem calculado F1, por isso nem recalculamos.



A iteração 8 podemos reaproveitar os cálculos de F2 e F1 para obtermos F3 $F(3) = 1 + 1 = 2$

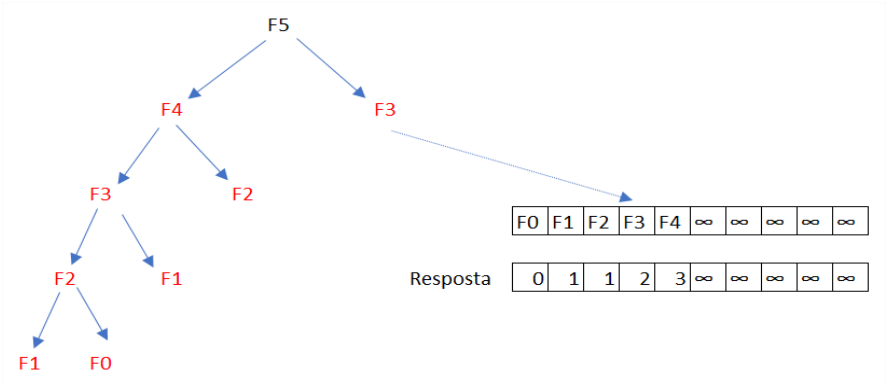


A iteração 9 verificamos na tabela que já tem calculado F2, por isso nem recalculamos

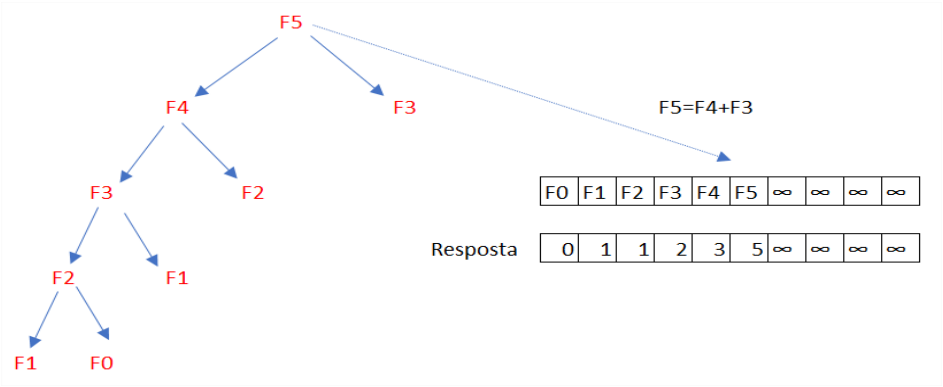


A iteração 10 podemos reaproveitar os cálculos de F3 e F2 para obtermos F4

$F(4) = 2 + 1 = 3$



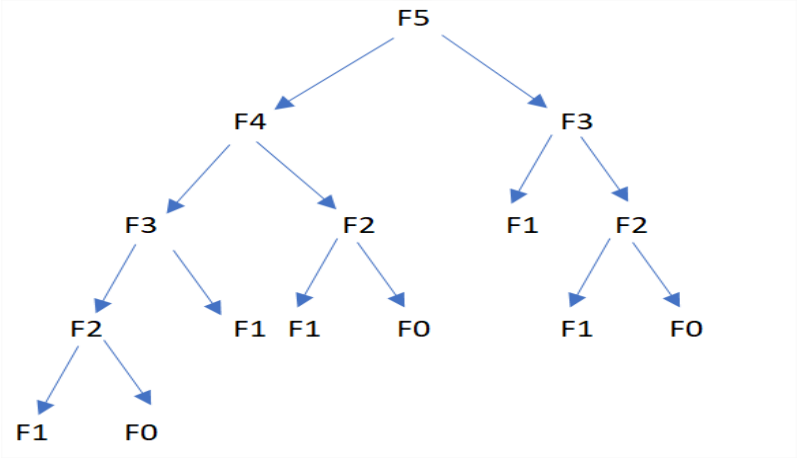
A iteração 11 verificamos na tabela que já tem calculado F3, por isso nem recalculamos



A iteração 12 podemos reaproveitar os cálculos de F4 e F3 para obtermos F5

$F(5) = 3 + 2 = 5$

Se fosse resolver de forma de recursiva, teríamos que recalcular todos os nós



4.5 Comparação de desempenho

A tabela 1 abaixo mostra o resulta de desempenho entre a técnica de programação.

Tabela 1:Teste de desempenho

iteração	resultado	número	Fibonacci programação dinâmica	Fibonacci Iterativo	Fibonacci Recurssivo
0	610	15	0	0	0
1	6765	20	0	0	0
2	75025	25	0	0	0
3	832040	30	0	0	12
4	9227465	35	0	0	142
5	102334155	40	0	2	1194
6	1134903170	45	0	1	12739
7	12586269025	50	0	1	140885
8	139583862445	55	0	1	1573611

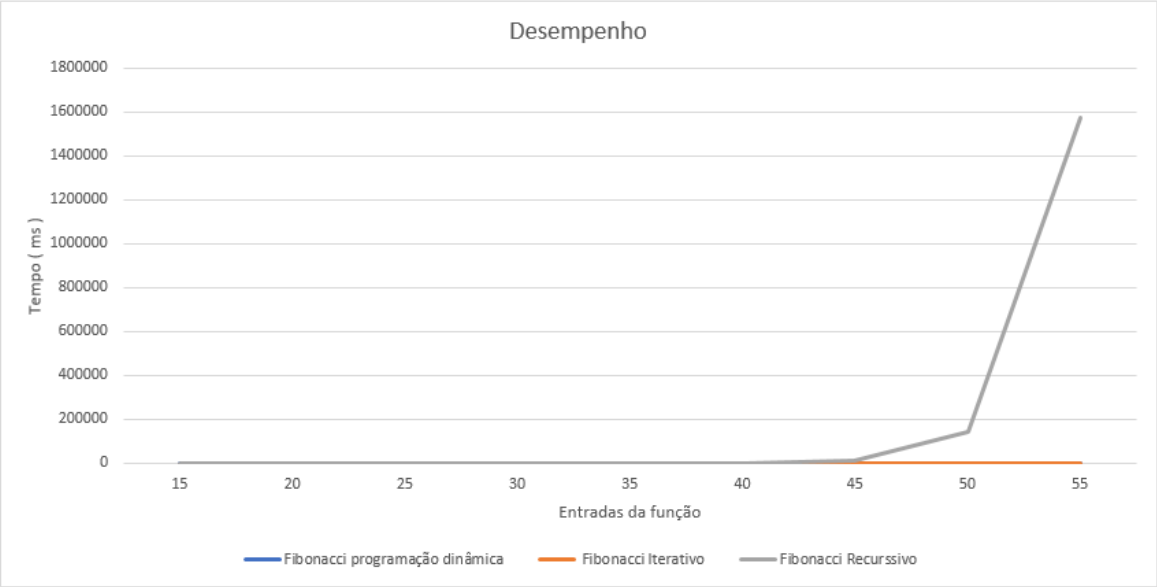


Figura 7:Desempenho dos algoritmos

REFERÊNCIAS

T. H. Cormen, C. E. Leiserson, R. L. Rivest e C. Stein, Introduction to Algorithms, McGraw-Hill, 2001, second edition.

Anexo: PROJETO EM C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Console;
using System.Diagnostics;

namespace ConsoleApplication1
{
    class Program
    {

        static public void combinar_item_na_muchila(int contar_tem, int
Num_elementos, int total_de_combinacoes, int[,] capacidade, int[,] codigo_gray,
int[] peso, int[] preco)
        {
            for (int j = 0; j < Num_elementos; j++)
            {//  faz toda a combinacoes de item  (muchila  0/1 )
                capacidade[contar_tem, 0] = capacidade[contar_tem, 0] +
codigo_gray[contar_tem, j] * peso[j];
                capacidade[contar_tem, 2] = capacidade[contar_tem, 2] +
codigo_gray[contar_tem, j] * preco[j];
                capacidade[contar_tem, 1] = contar_tem;
            }
            if (contar_tem < total_de_combinacoes)
            {
                contar_tem = contar_tem + 1;
                combinar_item_na_muchila(contar_tem, Num_elementos,
total_de_combinacoes, capacidade, codigo_gray, peso, preco);
            }
        }
//-------------------------------------------------------------------------------
        static public void combinar_item_na_muchila_dinamicamente(int
total_de_linhas, int[,] codigo_gray, int[,] capacidade, int[] peso, int[] preco, int
colunas)
        {
            int contador = 0, linhas = 0, linha_j = 0;
            for (int i = 1; i <= total_de_linhas; i++)///  verifica a  tabela
verdade
            {
                contador = contador + 1;
                if (contador >= total_de_linhas / 2 - 1)
                {
                    if (linhas < total_de_linhas / 2)
                    { linhas = linhas + 1; }
```

```
                }
                for (int j = 0; j < colunas; j++)        ///  verifica a  tabela
verdade
                {
                    if (contador < total_de_linhas / 2)    ///capacidade3[i,0]
                    {
                        capacidade[i, 0] = capacidade[i, 0] + codigo_gray[i, j] *
peso[j];
                        capacidade[i, 2] = capacidade[i, 2] + codigo_gray[i, j] *
preco[j];
                        capacidade[i, 1] = i;
                    }
                    else
                    {
                        capacidade[contador, 0] = codigo_gray[i, j] * peso[j] +
capacidade[i, 0];
                        capacidade[i, 2] = capacidade[i, 2] + codigo_gray[i, j] *
preco[j];
                        capacidade[i, 1] = i;
                    }
                }
            }
        }
///-------------------------------------------------------------------------------

        static public void Matriz_codigo_gray(double numero_de_elementos, int
colunas, int[,] codigo_gray)
        {
            int contador = 0;
            int contador2 = 0;
            int contador3 = 0;
            int linhas = (int)(Math.Pow(2, numero_de_elementos));
            int linha_j = (int)linhas;
            for (int i = 0; i < colunas; i++)
            {
                contador = (int)(Math.Pow(2, numero_de_elementos) / Math.Pow(2, i +
1));

                for (int j = 0; j < linhas * linhas; j++)
                {
                    if (contador2 < contador)
                    {
                        contador2 = contador2 + 1;
                        contador3 = contador3 + 1;
                        codigo_gray[contador3, i] = 0;
                    }
                    else
                    {
                        if (contador2 < 2 * contador)
                        {
                            contador2 = contador2 + 1;
                            contador3 = contador3 + 1;
                            codigo_gray[contador3, i] = 1;
                        }
                        else
                        {
                            contador2 = 0;
                        }
                    }
```

```
                    if (contador3 >= linhas)
                    {
                        j = (int)(linhas * linhas * linhas);
                        contador2 = 0;
                        contador3 = 0;
                    }
                }
            }
        }
//-------------------------------------------------------------------------------

        static public void imprimir_resultado(int total_de_linhas, int peso_total,
int[,] capacidade, int colunas, int[,] codigo_gray, char[] array1)
        {
            for (int i = 0; i <= total_de_linhas; i++)///  verifica a  tavela
verdade
            {
                if (capacidade[i, 0] > 0 && capacidade[i, 0] <= peso_total)
                {
                    Console.Write("Peso (" + (i + 1) + " ) = ");
                    for (int j = 0; j < colunas; j++)
                    {
                        Console.Write("   " + codigo_gray[capacidade[i, 1], j] + "."
+ array1[j]);
                    }
                    Console.Write(" = " + capacidade[i, 0] + " " + " (capacidade de
mochila = " + peso_total + " ) " + "  Max R$ " + capacidade[i, 2] + ",00 \n");
                }
            }

        }
        static public void ordenar_matriz(int total_de_linhas, int[,] capacidade)
        {
            int temp = 0;
            int temp2 = 0;
            int temp3 = 0;
            for (int i = 0; i < total_de_linhas; i++)// n
            {
                for (int percorre_vetor = 0; percorre_vetor < total_de_linhas;
percorre_vetor++)// n-1
                {
                    if (capacidade[percorre_vetor, 0] < capacidade[percorre_vetor +
1, 0])// troca
                    {
                        temp = capacidade[percorre_vetor + 1, 0];
                        temp2 = capacidade[percorre_vetor + 1, 1];
                        temp3 = capacidade[percorre_vetor + 1, 2];
                        capacidade[percorre_vetor + 1, 0] =
capacidade[percorre_vetor, 0];
                        capacidade[percorre_vetor + 1, 1] =
capacidade[percorre_vetor, 1];
                        capacidade[percorre_vetor + 1, 2] =
capacidade[percorre_vetor, 2];
                        capacidade[percorre_vetor, 0] = temp;
                        capacidade[percorre_vetor, 1] = temp2;
                        capacidade[percorre_vetor, 2] = temp3;
                    }
                }
            }
        }
```

```
///-------------------------------------------------------------------------------
        static public void Problema_da_mochila(int numero_de_elementos, int selecao)
        {
            int colunas = numero_de_elementos;
            double linhas = Math.Pow(2, numero_de_elementos);
            int total_de_linhas = (int)linhas;

            int[,] codigo_gray;
            codigo_gray = new int[total_de_linhas + 1, colunas + 1];
            int[,] capacidade;
            capacidade = new int[total_de_linhas + 1, 4];

            char[] array1 = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                'U', 'V', 'W', 'X', 'Y',
'Z','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t',
'u','v','w','x','y','z' };
            // seta rotulo no produto -

            Matriz_codigo_gray(numero_de_elementos, colunas, codigo_gray);

            int[] preco = new int[colunas * 5];
            int[] peso = new int[colunas * 5];
            int peso_total = 40;
            char[] produto = new char[colunas];

            if (selecao == 1)
            {
                Console.Write("Digite peso da muchila ");
                peso_total = Int32.Parse(Console.ReadLine());

                for (int i = 0; i < colunas; i++) //   entrada de  dados
                {
                    Console.Write("Produto " + array1[i] + " \n");
                    Console.Write("Digite peso    " + (i + 1) + "   =  ");
                    peso[i] = Int32.Parse(Console.ReadLine());

                    Console.Write("Digite preço   ");
                    preco[i] = Int32.Parse(Console.ReadLine());
                    Console.Write(" \n");
                }
            }
            else
            {
                peso[0] = 7; preco[0] = 25;
                peso[1] = 2; preco[1] = 14;
                peso[2] = 15; preco[2] = 42;
                peso[3] = 5; preco[3] = 71;
                peso[4] = 11; preco[4] = 151;
            }

            int contador10 = 1;

            if (selecao == 1)
            {
                combinar_item_na_muchila(contador10, colunas, total_de_linhas,
capacidade, codigo_gray, peso, preco);
            }
            else
            {
```

```
                combinar_item_na_muchila_dinamicamente(total_de_linhas, codigo_gray,
capacidade, peso, preco, colunas);
            }
            ordenar_matriz(total_de_linhas, capacidade);
            imprimir_resultado(total_de_linhas, peso_total, capacidade, colunas,
codigo_gray, array1);
        }
//--------------------------------------------------------------------------------
static public void linha_Producao_redefinida(int estacao, int linha, int
estacao_inicio, int[,] producao, int[,] percurso_antes, int[,] percurso)
        {  // linha de produção redefinida dinamicamente
            int total = percurso[4, estacao_inicio - 2]; //  pega  o  tempo anterior
            int linha_inicio = 5;   // seta a linha  onde contém  os dados  de
percursos
            percurso[linha_inicio, estacao_inicio] = 1;
            int frag = 0;


            for (int i = 0; i < linha - 1; i++)
            {
                for (int j = estacao_inicio - 1; j <= estacao; j++) // entrada de
dados
                {
                    int tempo = 1000000000;
                    int linha_de_montagem = 0;

                    for (int k = 0; k < linha; k++)      // entrada de  dados
                    {
                        if (producao[k, j] < tempo)
                        {
                            if (frag == 0)
                            {
                                if (0 == producao[0, j])
                                {
                                    tempo = producao[0, j];
                                    linha_de_montagem = k;

                                    Console.Write("tempo 0 " + tempo + "\n");
                                }
                                if (1 == producao[1, j])
                                {
                                    tempo = producao[1, j];
                                    linha_de_montagem = k;
                                    Console.Write("tempo 1 " + tempo + "\n");
                                }
                                frag = 1;
                            }
                            else
                            {
                                tempo = producao[k, j];
                                linha_de_montagem = k;

                            }
                        }
                    }
                    if (percurso[5, j] == 1 && j != 0)// voltar
                    {
                        total = total + tempo;
                        percurso_antes[0, j] = percurso[0, j];
                        percurso_antes[1, j] = percurso[1, j];
```

```
                    percurso_antes[3, j] = percurso[3, j];
                    percurso_antes[4, j] = total;
                }
                else
                {
                    total = total + tempo;
                    percurso_antes[0, j] = linha_de_montagem;
                    percurso_antes[1, j] = tempo;
                    percurso_antes[3, j] = j;
                    percurso_antes[4, j] = total;


                }
            }
        }
    }
//-----------------------------------------------------------------------------

    static public void Linha_de_producao(int estacao, int linha, int[,]
producao, int[,] percurso)
    {
        int total = 0;
        for (int i = 0; i < linha - 1; i++)
        {
            for (int j = 0; j <= estacao; j++)      // entrada de  dados
            {
                int tempo = 1000000000;
                int linha_de_montagem = 0;

                for (int k = 0; k < linha; k++)      // entrada de  dados
                {
                    if (producao[k, j] < tempo)
                    {
                        tempo = producao[k, j];
                        linha_de_montagem = k;
                    }
                }

                total = total + tempo;
                percurso[0, j] = linha_de_montagem;
                percurso[1, j] = tempo;
                if (j == 0)
                {
                    percurso[2, j] = linha_de_montagem;

                }
                else
                {
                    if (percurso[2, j] == percurso[2, j + 1])
                    {
                        percurso[2, j] = 0;
                    }
                    else
                    {
                        percurso[2, j] = 1;
                    }
                }

                percurso[3, j] = j;
                percurso[4, j] = total;
```

```
                    percurso[5, j] = 0;
                }
            }
        }
 //--------------------------------------------------------------------------------
        static public void mochila(int estacao, int linha, int[,] producao, int
selecao)
        {
            if (selecao == 0)
            {
                for (int i = 0; i < linha; i++) //   entrada de  dados
                {

                    for (int j = 0; j < estacao; j++)
                    {
                        Console.Write("linha " + (i + 1) + " estacao " + j + " \n");
                        producao[i, j] = Int32.Parse(Console.ReadLine());

                    }
                }

                for (int i = 0; i < linha; i++)
                {
                    for (int j = 0; j < estacao; j++)      // entrada de  dados
                    {
                        Console.Write("  " + producao[i, j]);

                    }
                    Console.Write("\n");
                }
            }
            else
            {
                producao[0, 0] = 1; producao[0, 3] = 2;
                producao[1, 0] = 12; producao[1, 3] = 25;
                producao[0, 1] = 1; producao[0, 4] = 1;
                producao[1, 1] = 16; producao[1, 4] = 5;
                producao[0, 2] = 22; producao[0, 5] = 37;
                producao[1, 2] = 1; producao[1, 5] = 30;

            }
        }
        //--------------------------------------------------------------------------
     static public void imprimir(int estacao, int linha, int[,] producao, int
selecao, int[,] percurso, int[,] percurso_antes)
        {
            Console.Write("\n\n   Linha de montagem com menor tempo       \n\n ");
            Console.Write("\r");
            for (int j = 0; j <= estacao; j++)      // entrada de  dados
            {
                        Console.Write("tempo     " + percurso[1, j] + "     linha
" + percurso[0, j] + "   estacao    " + percurso[3, j] + "    total     " +
percurso[4, j] + " Minutos    Redefinir  " + percurso[5, j] + "\n");
            }
            Console.Write(" ----------------------------------------------------\n");
            Console.Write("\n\n   Linha de montagem Redifenida            \n\n\t");
            Console.Write("\r");
            for (int j = 0; j <= estacao; j++)      // entrada de  dados
            {
```

```
                Console.Write("tempo     " + percurso_antes[1, j] + "     linha
" + percurso_antes[0, j] + "   estacao    " + percurso_antes[3, j] + "    total
" + percurso_antes[4, j] + "\n");
            }
        }

  static public void imput(int estacao, int linha, int[,] producao, int selecao)
   {
            for (int i = 0; i < linha; i++)
            {
                for (int j = 0; j <= estacao; j++)      // entrada de  dados
                {
                    Console.Write("S(" + (j) + ")-> " + producao[i, j] + "\t");
                }
                Console.Write("\n\n\r");
            }
     }
///-------------------------------------------------------------------------------
static public void Problema_da_linha_de_montagem(int estacao, int linha, int selecao)

        {  int[,] producao;
            producao = new int[linha + 1, estacao + 1];
            int[,] percurso;
            percurso = new int[6, estacao + 2];
            int[,] percurso_antes;
            percurso_antes = new int[6, estacao + 2];
            int total = 0;

            mochila(estacao,linha,producao,selecao);
            Linha_de_producao(estacao,linha,producao,percurso);

            imput(estacao,linha,producao,selecao);

            Console.Write("\n\n\n");
            int estacao_inicio;

            Console.Write(" Redefinir linha de montagem  ");
            estacao_inicio = Int32.Parse(Console.ReadLine()) + 1;


linha_Producao_redefinida(estacao,linha,estacao_inicio,producao,percurso_antes,
percurso);

          imprimir(estacao,linha,producao,selecao,percurso,percurso_antes);
        }

        public static Int64 fibonacci_Recursivo(Int64 n)
        {
          //  Console.Write("F1{" + n + "}  ");
            if (n == 0) return 0;
            else if (n == 1) return 1;
            else return fibonacci_Recursivo(n - 1) + fibonacci_Recursivo(n - 2);

        }

        public static Int64 fibonacci_Programacao_dinamica(Int64 n)
        {
            Int64 number = n;
```

```
    Int64[] Fib = new Int64[number + 1];
        Fib[0] = 0;
        Fib[1] = 1;

        for (Int64 i = 2; i <= number; i++)
        {
            Fib[i] = Fib[i - 2] + Fib[i - 1];
          //  Console.Write("F2{" + i + "}  " + Fib[i]+" ");
        }

        return Fib[number];
}
public static Int64 Fibonacci(Int64 n)
{
    Int64 a = 0;
    Int64 b = 1;
        for (int i = 0; i < n; i++)
        {
        Int64 temp = a;
            a = b;
            b = temp + b;

           // Console.Write("F3{" + i + "}  " + b+ " ");
        }
        return a;
}

static void Main(string[] args)
{
    var stopwatch = new Stopwatch();

    double[,] tempo_execucao;
    tempo_execucao = new double[20, 5];
    Int64 numero1 = 10;
    Int64 numero2 = 10;
    Int64 numero3 = 10;

    int tamanho = 0; ///  usa  5  para  o exemplo
    int programacao_dimanica = 0;
    int numero_estacao = 0;
    int linha_dinamica = 0;
    int contador = 0;
    long[] vector = new long[10];

    Console.Write("numero de elemento  ");
    tamanho = Int32.Parse(Console.ReadLine());
    Console.Write("Para exemplo (0) uso do usuario (1)  ");
    programacao_dimanica = Int32.Parse(Console.ReadLine());
    Problema_da_mochila(tamanho, programacao_dimanica);

    Console.Write("numero de estacao de montagem   ");
    numero_estacao = Int32.Parse(Console.ReadLine());
    Console.Write("Para exemplo (1) uso do usuario (0)  ");
    linha_dinamica = Int32.Parse(Console.ReadLine());

    if (linha_dinamica == 1)
    {
        numero_estacao = 5;
    }
    Problema_da_linha_de_montagem(numero_estacao, 2, linha_dinamica);
```

```
            for (Int64 i = 0; i < 6; i++)
            {

                numero1 = numero1+5;
                numero2 = numero2+5;
                numero3 = numero3+5;

                WriteLine("\n ---------------fibonacci   (  "+ numero3 + "  )-------
------------------------"+i+"---\n ");
                tempo_execucao[i,0] = i;
                WriteLine($"fibonacci_Programacao_dinamica :{
fibonacci_Programacao_dinamica(numero1)}");
                stopwatch.Start();  // tempo inicio
                stopwatch.Stop();  // tempo final
              //  WriteLine($"Tempo do fibonacci_Programacao_dinamica
:{stopwatch.ElapsedMilliseconds}");
                tempo_execucao[i, 1] = stopwatch.ElapsedMilliseconds;

                stopwatch.Reset();

                stopwatch.Start();  // tempo inicio
                WriteLine($"Fibonacci                       :{ Fibonacci(numero2)}");
                stopwatch.Stop();  // tempo final
             //   WriteLine($"Tempo do Fibonacci :{stopwatch.ElapsedMilliseconds}");
                tempo_execucao[i, 2] = stopwatch.ElapsedMilliseconds;
                stopwatch.Reset();

                stopwatch.Start();  // tempo inicio
                WriteLine($"fibonacci_Recursivo             :{
fibonacci_Recursivo(numero2)}");

                stopwatch.Stop();   // tempo final
             //   WriteLine($"Tempo do fibonacci_Recursivo
:{stopwatch.ElapsedMilliseconds}");
                tempo_execucao[i, 3] = stopwatch.ElapsedMilliseconds;
                stopwatch.Reset();

            }

            for (int i = 0; i < 9; i++)
            {
                WriteLine(" tempo (" + tempo_execucao[i, 0] + ") = " +
tempo_execucao[i, 1] + " " + tempo_execucao[i, 2] + " " + tempo_execucao[i, 3] +
"\n");
            }


            Console.ReadKey();
        }
    }
}
```