Programação Dinâmica

Resolução de Problemas

Programação Dinâmica: Resolver um problema resolvendo seus subproblemas e armazenado seus valores para o caso de precisar no futuro (o que melhora a eficiência).

1.Fibonacci

A sequência de FIbonacci é definida pela fórmula: F(n) = F(n-1) + F(n-2) onde F(0) = 0 e F(1) = 1.

Bottom-Up:

```c
#include <stdio.h>

int fibonacci(int n) {
 int fib[n+1];
 int i;
 fib[0] = 0;
 fib[1] = 1;
  for (i = 2; i <= n; i++){
   fib[i] = fib[i-1] + fib[i-2];
 }
 return fib[n];
}

int main(void) {
 int n;
 scanf("%d", &n);
 printf("O %dº elemento da sequencia de fibonacci e %d\n", n, fibonacci(n));
}
```

Top-Down:

```c
#include <stdio.h>

int fib_recursivo(int *fib, int n);

int fibonacci_top_down(int n){
 int fib[n+1];
 for(int i = 0; i <= n; i++){
   fib[i] = -1;
 }
 return fib_recursivo(fib, n);
}

int fib_recursivo(int *fib, int n){
```

```
 if(fib[n] >= 0){
   return fib[n];
 }
 if(n <= 1){
   fib[n] = n;
 }else{
   fib[n] = fib_recursivo(fib, n - 1) + fib_recursivo(fib, n - 2);
 }
 return fib[n];
}

int main() {
   int number;
   printf("Digite um numero inteiro para descobrir eu valor na sequencia de
Fibonacci:  ");
   scanf("%d", &number);
   printf("F(%d) = %d\n", number, fibonacci_top_down(number));
   return 0;
}
```

2.Corte da Haste

Dada uma haste de comprimento **n** polegadas e uma tabela de preços **p** onde **pi** para i **= 1, 2, ..., n,** determinar a receita máxima **rn** que se pode obter ao se cortar a haste e vender os seus pedaços. Os comprimentos dos pedaços são sempre números inteiros.

O código a seguir tem as duas partes TopDown e BottomUp além de uma versão extendida do BottomUp que retorna as posições de corte.

```
#include <stdio.h>

int max(int a, int b){
 if(a > b){
   return a;
 }else{
   return b;
 }
}
```

```c
int corte_haste_bottom_up(int *p, int n){
 int r[n+1];
 r[0] = 0;
 int q;
 for(int i = 1; i <= n; i++){
   q = -1;
   for(int j = 1; j <= i; j++){
     printf("max (%d, %d + %d)", q, p[j], r[i-j]);
     q = max(q, p[j] + r[i-j]);
     printf(" => %d\n",q);
   }
   r[i] = q;
   printf("r[%d] = %d\n", i, q);
 }
 return r[n];
}

int corte_haste_memo_aux(int *p, int n, int *r);

int corte_haste_top_down(int *p, int n){
  int r[n+1]; // cria vetor
  r[0] = 0; //caso inicial
  for(int i = 1; i <= n ; i++){
    r[i] = -1; // inicializa valores otimos
  }
  return corte_haste_memo_aux(p, n, r);
}

int corte_haste_memo_aux(int *p, int n, int *r){
  // Identifica que o foi calculado r[n] e o usa.
  if(r[n] >= 0)
    return r[n];
  // sera calculado um novo r[n]
 int q = -1;
 for(int i = 1; i <= n; i++){
   q = max(q, p[i] + corte_haste_memo_aux(p, n - i, r));
 }
 r[n] = q; // insere r[n] na tabela para ser usado em outras etapas
 printf("r[%d] = %d (top-down)\n", n, q);
 return q;
}
```

```c
int corte_haste_bottom_up_ext(int *p, int n, int *s){
 int r[n+1];
 r[0] = 0;
 int q;
 for(int i = 1; i <= n; i++){
   q = -1;
   for(int j = 1; j <= i; j++){
     if(q < p[j] + r[i-j]){
         q = p[j] + r[i-j];
         s[i] = j;
     }
   }
   r[i] = q;
 }
 return r[n];
}

int main() {

   int p[11] = {0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30};
   int number;

   printf("Digite o tamanho da haste (entre 1 e 10):  ");
   scanf("%d", &number);
   printf("Melhor valor para o uma haste de %d u.c. : R$ %d,00 \n", number,
corte_haste_top_down(p, number));

   // Usando corte_haste_bottom_up_ext para encontrar as posições de corte
   int s[number+1];
   corte_haste_bottom_up_ext(p, number, s);
   printf("Dado a haste de %d u.c Ela sera divida nas seguintes partes:\n",
number);
   if(0 == number - s[number]){
     printf("Nao corte, use-a inteira\n");
   }else{
     printf("Divide a haste em: ( | ");
     while(number > 0){
       printf("%d | ", s[number]);
       number = number - s[number];
     }
   }
   printf(")\n");

   return 0;
}
```

3.Maior Subsequência Comum (LCS)

Dadas duas sequências X = x1x2 … xm e Y = y1y2 … yn, encontrar uma subsequência comum a X e Y que seja o mais longa possível. Para as palavras : “noturno” e “mosquiteiro” a LCS pode ser “otro” ou “ouro”.

Bottom-Up:

```
int get_len_lcs(string & s1, string & s2){
    int len_s1 = s1.size(), len_s2 = s2.size();
    int mat[len_s1+1][len_s2+1];

    for(int i = 1; i <= len_s1; i++){
       mat[i][0] = 0;
    for(int i = 1; i < j; i++){
        q = max(q, p[i] + r[j-i]);
    }
    r[j] = q;
    }
    return r[n];
}
```