PROJETO E ANÁLISE DE ALGORITMOS - LABORATÓRIO 3

NOME: João Victor de Mesquita Cândido dos Santos **RA:** 102028

Explicação do Método:

A resolução do problema foi feita utilizando programação dinâmica com sobreposição de subproblemas onde o cálculo da solução por recursão implica no recálculo de subproblemas, assim sendo um problema é dividido em vários outros problemas e os subproblemas podem estar sendo resolvidos diversas vezes. A função que faz a solução desses subproblemas é chamada *Cut-Rod()* onde o número de chamadas dessa função é igual a:

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

Das versões da função *Cut-Rod()* existentes foi utilizada a versão *bottom-up* que resolve os problemas de pequena dimensão e guarda as soluções, assim sendo a solução de um problema é obtida combinando essa solução a soluções de problemas de menor dimensão, assim sendo resolve os menores problemas primeiros até chegar no maior problema.

Focando agora no nosso problema onde temos que encontrar o valor máximo de receita que é possível obter com a produção deste ano de café do produtor iniciamos o código alocando um vetor auxiliar que fará com que cada posição que é representada pelo tamanho do lote em número de sacas receba o valor a ser pago pelo lote, ou seja, se o tamanho do lote é 25 e o valor a ser pago pelo lote for 100 então nesse vetor auxiliar na posição 25 será salvo o valor 100.

A alocação desse vetor auxiliar pode ser vista abaixo onde, utilizamos *calloc* para além de alocar esse vetor dinamicamente, fazemos com que todas as posições recebam um valor inicial 0 para que não haja algum problema de serem armazenados valores aleatórios.

```
int *aux=calloc(101,sizeof(int));
```

Tendo feito isso é então realizada a leitura de um valor P representando o número de sacas de café especial produzidas no ano e N que indica o número de possíveis compradores como pode ser visto abaixo.

```
scanf("%d %d",&P,&N);
```

Após isso entramos em um laço em que considerando o número de compradores fazemos a leitura do tamanho do lote em número de sacas e o valor a ser pago pelo lote para cada comprador.

Tendo feito as leituras precisamos verificar se naquela posição já não há algum valor armazenado, pois pode ocorrer o caso em que há dois compradores querendo comprar um mesmo lote, caso isso ocorra é sempre vendido para o comprador que está disposto a pagar mais, sendo assim chamamos uma função

que verifica qual é o valor máximo, se é o valor anterior ou o que será armazenado agora.

Após isso o valor a ser pago pelo lote é salvo na posição referente ao tamanho do lote no vetor auxiliar como pode ser visto abaixo.

```
for(k=1;k<=N;k++)  {
      scanf("%d",&p);
      scanf("%d",&n);
     x=max(aux[p],n);
       aux[p]=x; }
```

Após isso chamamos a função *BottomupCutRod* que irá resolver nosso problema em subproblemas e retornar a receita máxima que será salva em uma variável, além disso após o uso do vetor auxiliar é liberado o espaço na memória que este ocupava como pode ser visto abaixo.

```
f=BottomupCutRod(aux,P);
      printf("%d",f);
        free(aux);
```

A função *BottomupCutRod* recebe como parâmetro todo nosso vetor auxiliar e o valor de sacas de café especial, assim sendo esse valor será um limitante para a divisão em subproblemas assim sendo se o número de sacas é 30 a divisão será feita do menor para o maior subproblema até chegar no limitante 30.

```
int BottomupCutRod(int p[],int n)
```

O vetor que irá receber os valores dos subproblemas foi alocado semelhante ao vetor auxiliar da funçao *main()* como pode ser visto abaixo.

```
int *r=calloc((n+1),sizeof(int));
```

Assim sendo para cada uma das posições do vetor r que é limitado pelo tamanho do número de sacas, é realizada a divisão em subproblemas onde temos uma variável que vai sempre armazenar os valores do maior valor de sacas do nosso vetor r que em cada posição está armazenado o valor a ser pago pelo respectivo lote, então indo de dois em dois, ou seja verificando o valor atual com o valor anterior iremos obter uma receita máxima que será armazenada na posição atual do vetor r até que chegue no valor limitante que é a última posição, ou seja, os subproblemas de encontrar a receita máxima foram resolvidos desde da menor posição do vetor para a maior posição do vetor. Por fim retornamos para a função *main()* o valor final do problema que está armazenada na última posição, a posição n obtendo assim o valor máximo de receita e liberamos o espaço na memória utilizado pelo nosso vetor.

```
for (j=1;j<=n+1;j++)
{
```

```
        q = -1;
        for(i=1;i<=j;i++)
            q = max(q,p[i]+r[j-i]);
            r[j] = q;
    }
    return r[n];
    free(r);
}
```

Por fim vale citar o funcionamento simples da função de máximo utilizado no código que consiste nada mais em uma condição que retornará o valor do maior inteiro lido como pode ser visto abaixo.

```
int max(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

Análise de Complexidade

```
int max(int x,int y)
{
    if(x>y)
        return x; //Custo O(1)
    else
        return y; //Custo O(1)
}

int BottomupCutRod(int p[],int n){
    int q,k,i,j;
    int *r=calloc((n+1),sizeof(int)); //Custo O(1)
    for(k=0;k<=n;k++)//Custo O(n)
    {
        r[k]=0;
    }
    for (j=1;j<=n;j++)// Custo O(n)
    {
        q = -1;
        for(i=1;i<=j;i++) // Custo O(n²)
            q = max(q,p[i]+r[j-i]); // Custo O(1)*O(n²) = O(n²)
            r[j] = q; //Custo O(1)
    }
    return r[n]; //O(1)
    free(r); //O(1)
}
```

```c
int main()
{
  int P,N,n,p,f=0,i,k,x;
  int *aux=calloc(101,sizeof(int)); // Custo dessa linha é O(1)
  scanf("%d %d",&P,&N);
  for(k=1;k<=N;k++)  //Custo dessa linha é O(N)
  {
     scanf("%d",&p);
     scanf("%d",&n);
     x=max(aux[p],n);  //O(N)*O(1)
     aux[p]=x;  //O(N)*O(1)

  }
 f=BottomupCutRod(aux,P); //O(1)
 printf("%d",f);
 free(aux); //O(1)
}
```

Analisando o código e fazendo a somatória do custo de todas as linhas fazendo o somatório cujo resultado é: $2O(1)+O(1)+O(n)+O(n^2)+3O(1)+O(1)+O(N)+2O(1)$.Assim foi possível se obter que no final por predominância a complexidade do algoritmo será de $O(n^2)$, sendo n o número de sacas e café especial produzias neste ano, ou seja, no pior caso o algoritmo irá solucionar o problema com uma **complexidade quadrática**.