MC458 — PA2

Professor: Pedro J. de Rezende

Leonardo Rodrigues Marques 178610

Código - Completo
Relatório - Bom
Nota total - 1.0

1)

Um problema possui sub-estrutura ótima se existem soluções ótimas para seus sub-problemas. Em primeira instância, começamos com a base, ou seja, quando i = 0. Nessa instância, o custo mínimo é o estado atual, portanto basta copiar o valores dos custos da matriz *arr(i, j)*. A partir da base, vamos buscar pelos custos mínimos. A cada instância do problema, estamos sempre tentando minimizar o custo, portanto buscamos pela solução ótima daquela instância, e recorrentemente, construímos a solução final mais otimizada a partir dos sub-problemas testados.

$$arr_c(i,j) = \begin{cases} arr(i,j), & \text{se } i = 0 \\ arr(i,j) + min\{arr_c(i-1,j), arr_c(i-1,j+1)\} & \text{se } j = 1 \\ arr(i,j) + min\{arr_c(i-1,j-1), arr_c(i-1,j)\} & \text{se } j = m \\ arr(i,j) + min\{arr_c(i-1,j-1), arr_c(i-1,j), arr_c(i-1,j+1)\} & \text{se} j \neq 1, m \text{ e } i \neq 0 \end{cases}$$

Observando a fórmula de recorrência, observamos alguns casos. Considere n x m, o tamanho da matriz utilizada. Quando estamos na primeira (i = 0), temos o caso base, ou seja, o estado atual, portanto baste apenas copiar os valores da matriz de custos *arr(i, j)* para a matriz de custos dos percursos $arr_c(i,j)$. No segundo (j = 1) e terceiro casos (j = m), devido aos limites estrutura de dados utilizada, temos que limitar o escopo de busca dos percursos a esquerda e direita, respectivamente, por isso a minimização ocorre com apenas dois termos. E o quarto e último caso , corresponde ao problema intrínseco, ou seja, a busca pelo percurso mínimo utilizando as todas circunstâncias do problema (acima central, acima direita e acima esquerda), por isso a minimização ocorre em três termos.

2)

Usando a ideia do percurso mínimo ministrado em aula usando programação dinâmica e trabalhado em prova, foi necessário apenas adaptar o problema ao tipo do modelo requisitado. Abaixo, o algoritmo do projeto em C:

```
int findMinCost(int n, int m, int **arr, int **arr_c){

    int i, j;

    for (i = 0; i < n; i++){
        for (j = 0; j < m; j++){
            if ( i == 0){
                arr_c[i][j] = arr[i][j];
            } else if ( j == 0){
                arr_c[i][j] = arr[i][j] +
                min_2(arr_c[i - 1][j], arr_c[i - 1][j + 1]);
            } else if ( j == m - 1){
                arr_c[i][j] = arr[i][j] +
                min_2(arr_cost[i - 1][j], arr_c[i - 1][j - 1]);
            } else {
                arr_c[i][j] = arr[i][j] +
                min_3(arr_c[i - 1][j - 1],arr_c[i - 1][j], arr_c[i - 1][j + 1]);
            }
        }
    }

    int cost = INT_MAX;
    for (j = 0; j < m; j++){
        if (arr_cost[n - 1][j] < cost){
            cost = arr_cost[n - 1][j];
        }
    }
    return cost;
}
```

3)

Considere uma matriz de tamanho n x m, e que m $\in$ O(n). Observe que na linha 5, temos que i vai de 0 até n, executando n vezes, portanto temos complexidade O(n). A linha 6 é interna a linha 5, portanto para cada valor de i, temos que j vai de 0 até m, executando m vezes, e como m $\in$ O(n), portanto temos complexidade O(n). Concluímos então que o algoritmo executa em complexidade de tempo da ordem de $O(n) \cdot O(n) = O(n^2)$. Data a matriz n x m, complexidade de espaço é de ordem $O(n) \cdot O(m)$, portanto $O(n^2)$.