Análise de Algoritmos - Trabalho 1

Clara de Mattos Szwarcman - 1310351
Lucas Ribeiro Borges -
Guilherme Simas Abinader -

1 - Controle de Qualidade na Produção de Frascos de Vidro

1)

A altura será dividida em raiz de n intervalos, aonde cada intervalo possui raiz de n degraus. O primeiro frasco será jogado de raiz de n em raiz de n degraus. Quando o frasco quebrar, jogaremos o segundo frasco a partir do início desse intervalo de degrau em degrau até que ele quebre.

Pseudo Código:

$Degrau\_2\_frascos(x, n)$

$raiz\_n = sqrt(n);$

$for\ i = 0;\ i < n;\ i+ = raiz\_n$

$if\ i >= x$

$for\ j = i - raiz_n;\ j < i;\ j++$

$if\ j == x$

$return\ j;$

Quando o primeiro frasco quebrar teremos um intervalo de tamanho $\sqrt{n}$ que com certeza contém a altura em que o frasco quebra, visto que se o frasco não quebrou no degrau anterior ao início do intervalo, ele também não quebra em nenhum dos degraus abaixo dele. Assim, ao percorrermos o intervalo de um em um, encontraremos a altura x.

No pior dos casos, o frasco quebra no último degrau, portanto, o primeiro frasco será jogado de todos os intervalos ($\sqrt{n}$ vezes.) Como ele quebrou no último degrau, será conferido se ele quebra em algum degrau pertencente ao último intervalo. Dessa maneira, o segundo frasco será jogado em cada degrau do intervalo ($\sqrt{n}$ vezes), até finalmente quebrar no último degrau. Sendo assim, foi jogado $2 * \sqrt{n}$.

$$O(2 * \sqrt{n}) = O(\sqrt{n})$$

2)

Tendo 3 frascos:

O primeiro frasco será jogado em intervalos de $n^{\frac{2}{3}}$ até quebrar. O segundo frasco será jogado em intervalos de $n^{\frac{1}{3}}$, no intervalo de $n^{\frac{2}{3}}$ encontrado. O terceiro frasco será jogado de degrau em degrau no intervalo de $n^{\frac{1}{3}}$ encontrado.

Tendo 4 frascos:

O primeiro frasco será jogado em intervalos de $n^{\frac{3}{4}}$ até quebrar. O segundo frasco será jogado em intervalos de $n^{\frac{2}{4}}$, no intervalo de $n^{\frac{3}{4}}$ encontrado. O terceiro frasco será jogado em intervalos de $n^{\frac{1}{4}}$, no intervalo de $n^{\frac{2}{4}}$ encontrado. O quarto frasco será jogado de degrau em degrau no intervalo de $n^{\frac{1}{4}}$ encontrado.

Tendo k frascos:

A altura ($(\sqrt[k]{n})^k$ degraus)é dividida em $\sqrt[k]{n}$ intervalos de tamanho $(\sqrt[k]{n})^{k-1}$. Jogamos o primeiro frasco em intervalos de $(\sqrt[k]{n})^{k-1}$ degraus. Quando o frasco quebrar, o último intervalo ($(\sqrt[k]{n})^{k-1}$ degraus) será dividido em $\sqrt[k]{n}$ intervalos de tamanho $(\sqrt[k]{n})^{k-2}$ degraus e o segundo frasco será jogado em intervalos de $(\sqrt[k]{n})^{k-2}$ degraus. Isso ocorrerá sucessivamente para todos os k frascos. No frasco k teremos um intervalo de tamanho $(\sqrt[k]{n})^{k-(k-1)}$, que é igual a $\sqrt[k]{n}$. O frasco será jogado em intervalos de $(\sqrt[k]{n})^{k-k}$, ou seja, de degrau em degrau, até quebrar.

Pseudo Código:

$Degrau\_k\_frascos(x, n, k)$

$raiz\_kesima = raiz(n, k)$

$inicio = 0;$

$fim = n;$

$incremento = pow(raiz\_kesima, k-1)$

$for\ i = 0;\ i < k;\ i++$

$for\ j = inicio;\ j < fim;\ j+ = incremento$

$if\ j >= x$

$if\ incremento == 1$

$return\ j;$

$inicio = j - incremento;$

$fim = j;$

$incremento = incremento/raiz\_kesima;$

$break;$

Segue a premissa do primeiro, aonde sempre se tem certeza do intervalo em que ocorre a quebra, porém com mais frascos para serem utilizados. Portanto, podemos inicialmente dividir a altura em intervalos maiores com mais subdivisões, assim postergando a procura de um em um, que será feita em um intervalo menor.

Para cada frasco estamos realizando no máximo $\sqrt[k]{n}$ testes, visto que para o frasco i temos um espaço de $(\sqrt[k]{n})^{k-(i-1)}$ degraus e o jogaremos em intervalos de $(\sqrt[k]{n})^{k-i}$ degraus. Como temos k frascos, isso será realizado k vezes. Assim, o número total de quedas será $k * \sqrt[k]{n}$.

$$O(k * \sqrt[k]{n})$$

Se k, é um número fixo, a complexidade será O($\sqrt[k]{n}$), como provado anteriormente.

3)

A menor complexidade assintótica possível é de $O(logn)$.

O algoritmo realiza uma busca binária ao longo da escada, jogando um frasco a cada comparação, se o frasco quebra, busca-se na metade inferior, do contrário busca-se na metade superior. Quando o intervalo é de 1 degrau, podemos garantir que encontramos a altura correta.

Pseudo Código:

$Degrau\_logn\_frascos(x, n)$

$busca\_binaria(x, n)$

2. Problema da Mochila Fracionária (pode-se colocar parte de um objeto na mochila)

1.a)

Os objetos são ordenados por seu valor por peso com merge sort. São então adicionados a mochila de um em um começando pelo com maior valor por peso, até a capacidade ser atingida.

Pseudo Código:

```
struct objeto
    int valor;
    int peso;
    float densidade;
    int indice;
mochila_frac1(w, v, n, W)
    for i = 0; i < n; i++
        objetos[i] = cria_objeto(v[i], w[i], i);
        objetos_selecionados[i] = 0;
    mergesort_densidade(objetos, n);
    sum_peso = 0;
    for j = n - 1; j >= 0; j--
        if sum_peso == W
            return objetos_selecionados;
        indice = objetos[j]-> indice;
        peso = objetos[i]-> peso;
        if peso + sum_peso < W
            objetos_selecionados[indice] = peso;
            sum_peso+ = peso;
        else
            objetos_selecionados[indice] = W - sum_peso;
            sum_peso+ = W - sum_peso;
    return objetos_selecionados;
```

Com os elementos ornados por valor por peso, podemos facilmente sempre escolher quais resultarão em um maior valor na mochila.

Inicializar e percorrer os vetores é realizado em $O(n)$. A ordenação com mergesort é realizada em $O(n\ logn)$.

$$O(c*n + n \log n) = O(n \log n)$$

1.b)

Encontra-se a mediana do valor por peso e particiona-se o vetor.

- Se a metade de maior valor cabe na mochila, todos os objetos são colocados na mochila e realiza-se o algoritmo na metade de menos valor.
- Se a metade de maior valor não cabe na mochila, realiza-se o algoritmo na metade de maior valor.
- Se a metade de maior valor possui apenas um objeto, coloca-se a fração dele que cabe na mochila.

Pseudo Código:

struct objeto

$int\ valor;$

$int\ peso;$

$float\ densidade;$

$int\ indice;$

$mochila\_frac2(w, v, n, W)$

$for\ i = 0;\ i < n;\ i++$

$objetos[i] = cria\_objeto(v[i], w[i], i);$

$objetos\_selecionados[i] = 0;$

$mochila\_frac2\_rec(objetos, W, 0, n, objetos\_selecionados);$

$return\ objetos\_selecionados;$

$mochila\_frac2\_rec(objetos, W, ini, fim, objetos\_selecionados)$

$if\ ini > fim$

$return;$

$if\ ini == fim$

$indice = objetos[ini]- > indice;$

$objetos_selecionados[indice] = W;$

$return;$

$meio = ini + fim/2$

$k = kesima\_densidade(objetos, meio);$

$part\_inv\_densidade(ini, fim, objetos, k);$

$soma = somar\_peso(objetos, ini, meio);$

$if\ soma > W$

$mochila\_frac2\_rec(objetos, W, ini, meio, objetos\_selecionados);$

$else$

$for\ i = ini;\ i < meio;\ i++;$

$indice = objetos[i]- > indice;$

$peso = objetos[i]- > peso;$

$objetos\_selecionados[indice] = peso;$

$mochila\_frac2\_rec(objetos, W - soma, meio, fim, objetos\_selecionados)$

Com o particionamento dos objetos pela mediana do valor por peso, asseguramos que sempre que uma metade é colocada na mochila, esta é a metade de maior valor.

Achar a mediana, particionar e somar são realizados em $O(n)$. Assim, temos a seguinte relação de recorrência:

$$f(n) \leqslant \begin{cases} c,\ n = 1 \\ f(\frac{n}{2}) + c'n,\ n > 1 \end{cases}$$

$Pelo\ Teorema\ Mestre:$

$a = 1$
$b = 2$
$k = 1$
$a < b^k$

$$O(n^k) = O(n)$$

1.c)

Pseudo Código:

$struct\ objeto$

$int\ valor;$

$int\ peso;$

$float\ densidade;$

$int\ indice;$

$mochila\_frac3(w, v, n, W)$

$for\ i = 0;\ i < n;\ i++$

$objetos[i] = cria\_objeto(v[i], w[i], i);$

$objetos\_selecionados[i] = 0;$

$mochila\_frac3\_rec(objetos, W, 0, n, objetos\_selecionados);$

$return\ objetos\_selecionados;$

$mochila\_frac3\_rec(objetos, W, ini, fim, objetos\_selecionados)$

$if\ ini > fim$

$return;$

$if\ ini == fim$

$indice = objetos[ini]- > indice;$

$objetos_selecionados[indice] = W;$

$return;$

$meio = ini + fim/2$

$valor_pivot = media_densidade(objetos, ini, fim);$

$part\_inv\_densidade(ini, fim, objetos, valor_pivot);$

$soma = somar\_peso(objetos, ini, meio);$

$if\ soma > W$

$mochila\_frac3\_rec(objetos, W, ini, meio, objetos\_selecionados);$

$else$

$for\ i = ini;\ i < meio;\ i++;$

$indice = objetos[i]->indice;$

$peso = objetos[i]->peso;$

$objetos\_selecionados[indice] = peso;$

$mochila\_frac3\_rec(objetos, W - soma, meio, fim, objetos\_selecionados)$

Agora podemos ter um caso de particionamento desbalanceado, onde sempre um lado tem 1 elemento e o outro tem n-1 elementos. Se todos elementos menos uma fração do último cabem na mochila, todos serão percorridos. Assim, teremos intâncias de $n-1, n-2, ..., 1$, elementos. Portanto a soma de todas as iterações é igual a $\frac{n-1(n-2)}{2}$.

$$O(\frac{n-1(n-2)}{2}) = O(n^2)$$