AED2 - Aula 13
Problema da seleção

"Podemos fazer melhor?"
- mote do projetista de algoritmos

Nesta aula vamos estudar um problema básico e descobrir como reaproveitar ideias centrais de alguns algoritmos de ordenação para desenvolver diversas soluções interessantes para ele.

Problema da seleção

Definições:

- a ordem de um elemento é uma medida da grandeza dele
    - em relação aos seus pares.
- Assim, se a ordem de um elemento é k
    - então existem k elementos de valor menor que o dele.
- Dado um vetor v de tamanho n e um inteiro k em [0, n)
    - no problema da seleção queremos o valor do elemento de ordem k

Exemplos:

- 3 2 5 4 1 e k = 3
    - Elemento de ordem 3 é 4
- 1 2 3 4 5 e k = 3
    - Elemento de ordem 3 é 4
- 5 4 3 2 1 e k = 3
    - Elemento de ordem 3 é 4

Curiosidades:

- Na permutação ordenada do vetor v
    - o elemento de ordem k ocupa a k-ésima posição.
- Note que podemos definir ordem começando em 0 ou em 1.
    - Escolhi usar ela começando em 0, para combinar com nossos vetores.
    - Assim, o elemento de ordem k ocupa a posição v[k] se v for ordenado.
- Perceba que o problema do mínimo e do máximo são casos particulares
    - do problema da seleção.
    - Mínimo corresponde ao elemento de ordem 0.
    - Máximo corresponde ao elemento de ordem n - 1.
- Observe que o problema da seleção é trivial se v estiver ordenado,
    - ou se ordenarmos ele.
        - Qual seria a eficiência dessa abordagem?
- Será que conseguimos resolver o problema sem usar esta abordagem?

Algoritmos:

```
int selecao1(int v[], int n, int k)
{
   int i, j, ind_min, aux;
   for (i = 0; i <= k; i++)
   {
       ind_min = i;
       for (j = i + 1; j < n; j++)
           if (v[j] < v[ind_min])
               ind_min = j;
       troca(&v[i], &v[ind_min]);
   }
   return v[k];
}
```

- Invariante e corretude: v[0 .. i - 1] está ordenado e é <= que v[i .. n - 1]
- Eficiência de tempo: O(k n)
- Eficiência de espaço: O(1) espaço adicional

```
int selecao2(int v[], int n, int k)
{
   int i, m = n;
   for (i = n / 2; i >= 0; i--)
       desceHeap(v, n, i);
   for (m = n - 1; m >= k; m--)
   {
       troca(&v[0], &v[m]);
       desceHeap(v, m, 0);
   }
   return v[k];
}
```

- Invariante e corretude:
    - v[m + 1 .. n - 1] está ordenado e é >= que v[0 .. m],
        - que é um heap de máximo
- Eficiência de tempo: O(n + (n - k) lg n)
- Eficiência de espaço: O(1) espaço adicional

```
// p indica a primeira posicao e r a ultima
int selecao3(int v[], int p, int r, int k)
{
   int j;
   j = separa(v, p, r);
   if (k == j)
       return v[j];
   if (k < j)
       return selecao3(v, p, j - 1, k);
   // if (k > j)
```

```
    return selecao3(v, j + 1, r, k);
}
```

- Eficiência de tempo: O(n^2)
- Eficiência de espaço: O(n) espaço adicional

```
// p indica a primeira posicao e r a ultima
int selecao4(int v[], int p, int r, int k)
{
    int desl, j;
    desl = (int)(((double)rand() / (RAND_MAX + 1)) * (double)(r - p + 1));
    troca(&v[p + desl], &v[r]);
    j = separa(v, p, r);
    if (k == j)
        return v[j];
    if (k < j)
        return selecao4(v, p, j - 1, k);
    // if (k > j)
    return selecao4(v, j + 1, r, k);
}
```

- Eficiência de tempo esperado: O(n)
- Eficiência de espaço esperado: O(lg n) espaço adicional

```
// p indica a primeira posicao e r a ultima
int selecao5(int v[], int p, int r, int k)
{
    int desl, j;
    while (1)
    {
        desl = (int)(((double)rand() / (RAND_MAX + 1)) * (double)(r - p + 1));
        troca(&v[p + desl], &v[r]);
        j = separa(v, p, r);
        if (k == j)
            return v[j];
        if (k < j)
            r = j - 1;
        else // if (k > j)
            p = j + 1;
    }
}
```

- Invariante e corretude:
  - v[0 .. p - 1] <= v[p .. r] <= v[r + 1 .. n - 1]
  - depois do separa() v[j] corresponde ao j-ésimo elemento
- Eficiência de tempo esperado: O(n)
- Eficiência de espaço: O(1) espaço adicional

```
// p indica a primeira posicao e r a ultima
int selecao6(int v[], int n, int k)
```

```
{
   int desl, j;
   int p = 0;
   int r = n - 1;
   while (k != j)
   {
       desl = (int)(((double)rand() / (RAND_MAX + 1)) * (double)(r - p + 1));
       troca(&v[p + desl], &v[r]);
       j = separa(v, p, r);
       if (k < j)
           r = j - 1;
       else // if (k > j)
           p = j + 1;
   }
   return v[j];
}
```

- Invariante e corretude:
  - v[0 .. p - 1] <= v[p .. r] <= v[r + 1 .. n - 1]
  - depois do separa() v[j] corresponde ao j-ésimo elemento
- Eficiência de tempo esperado: O(n)
- Eficiência de espaço: O(1) espaço adicional