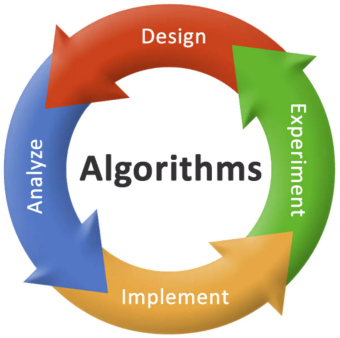
AED2 - Aula 09
Projeto e análise de algoritmos, segmento de soma máxima

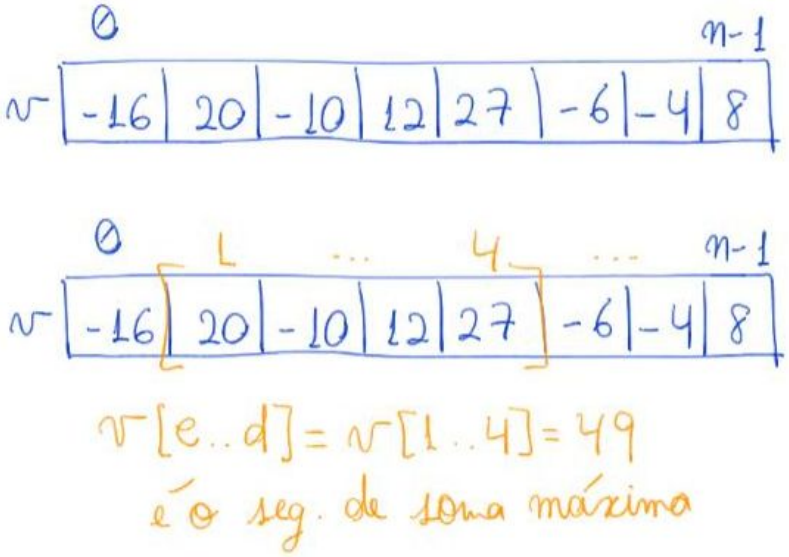
"Podemos fazer melhor?"
- mote do projetista de algoritmos



Compreensão, verificação da corretude e análise da eficiência de algoritmos iterativos para o problema do segmento de soma máxima.

Problema do segmento de soma máxima:

- Dado um vetor v[0 .. n - 1],
  - um segmento de v é qualquer subvetor de v da forma
    - v[e .. d], com 0 <= e <= d < n.
  - Se e > d então o segmento é vazio.
- Considerando que os elementos de v são inteiros,
  - a soma de um segmento corresponde à soma dos seus elementos.
- Assim, desejamos encontrar um segmento de soma máxima
  - i.e., um segmento cuja soma dos elementos seja >=
    - que a soma dos elementos de qualquer outro segmento de v.
- Exemplo:



Observem que, um segmento é determinado pelos seus extremos (e, d).

- Portanto, dado um vetor de tamanho v, existem
  - (n escolhe 2) = n (n - 1) / 2 segmentos diferentes.

Além disso, podemos calcular o valor de um segmento

- em tempo linear no tamanho do segmento,
  - i.e., proporcional a (d - e),

- usando a seguinte rotina

```
// soma os valores em v[e .. d] e devolve em *psoma
void somaSeg(int v[], int e, int d, int *psoma)
{
    int i;
    *psoma = 0;
    for (i = e; i <= d; i++)
        *psoma += v[i];
}
```

Corretude de algoritmos iterativos:

- Enunciar relações invariantes que valem ao longo das iterações
  - *psoma é a soma dos elementos em v[e .. i - 1].
- Mostrar que as relações valem no início da primeira iteração
  - No início *psoma = 0 e v[e .. i - 1] = v[e .. e - 1] é vazio.
    - Portanto, o resultado vale trivialmente.
- Mostrar que, se as relações valem no início de uma iteração qualquer,
  - então elas continuam valendo no início da próxima iteração.
    - No início da i-ésima iteração *psoma = v[e] + … + v[i - 1].
    - Depois da instrução *psoma += v[i] temos
      - *psoma = v[e] + … + v[i - 1] + v[i].
    - Ao final da iteração, ocorre i++.
    - Portanto, no início da próxima iteração
      - *psoma = v[e] + … + v[i - 1].
- Verificar que, quando os laços terminam,
  - os invariantes implicam a corretude do algoritmo.
    - Quando o laço termina temos i = d + 1.
    - Pelo invariante, *psoma = v[e] + … + v[i - 1] = v[e] + … + v[d].
    - Portanto, o algoritmo devolve a soma do segmento v[e .. d].

Eficiência de tempo:

- Número de operações é proporcional a (d - e), i.e., O(d - e). Por que?

Eficiência de espaço:

- Quantidade de memória auxiliar utilizada é constante em relação à entrada,
  - i.e., O(1). Por que?

Combinando as ideias expostas anteriormente, podemos

- verificar a soma de cada um dos (n escolhe 2) segmentos e pegar o maior.
- Esta ideia é implementada no nosso primeiro algoritmo

```
// encontra o segmento de soma máxima em v[0 .. n - 1] e
// devolve seus limites em *pe, *pd e valor em *psMax
void segMax3(int v[], int n, int *pe, int *pd, int *psMax)
{
    int i, j, k, sAux;
    *psMax = 0;
```

```
    *pe = *pd = -1;
    for (i = 0; i < n; i++) // i é o início do segmento corrente
        for (j = i; j < n; j++) // j é o final de tal segmento
        {
            // somaSeg(v, i, j, &sAux);
            sAux = 0;
            for (k = i; k <= j; k++)
                sAux += v[k];
            if (sAux > *psMax)
            {
                *psMax = sAux;
                *pe = i;
                *pd = j;
            }
        }
}
```

Invariantes e corretude:

- Observe que, na i-ésima iteração do laço externo
  - calculamos a soma de todos os segmentos que começam em i.
- De modo semelhante, na j-ésima iteração do segundo laço
  - calculamos a soma dos segmentos que terminam em j.
- O laço mais interno é responsável por somar os valores em v[i .. j].
- Os principais invariantes do laço externo são
  - v[*pe .. *pd] é um segmento de soma máxima de v com *pe < i.
  - *psMax = v[*pe] + … + v[*pd].

Eficiência de tempo:

- Número de operações no pior caso da ordem de n^3, i.e., O(n^3),
  - por conta dos três laços aninhados,
    - cada um podendo ter tamanho da ordem de n.

Eficiência de espaço:

- Memória auxiliar utilizada é constante em relação à entrada, i.e., O(1).

Curiosidade:

- Podemos trocar as linhas

```
            sAux = 0;
            for (k = i; k <= j; k++)
                sAux += v[k];
```

- pela seguinte chamada da função somaSeg.

```
            somaSeg(v, i, j, &sAux);
```

- O algoritmo continua correto? Isso afeta a eficiência do algoritmo?

Observando o problema, podemos perceber que

- a soma de um segmento v[e .. d] corresponde
    - à soma do seguimento v[e .. d - 1] com v[d].
- Analisando segMax3 atentos à essa observação, percebemos que
    - ele recalcula muitas vezes a soma dos mesmos segmentos.
- Eliminando esses re-cálculos temos nosso segundo algoritmo.

```c
// encontra o segmento de soma máxima em v[0 .. n - 1] e
// devolve seus limites em *pe, *pd e valor em *psMax
void segMax2(int v[], int n, int *pe, int *pd, int *psMax)
{
    int i, j, sAux;
    *psMax = 0;
    *pe = *pd = -1;
    for (i = 0; i < n; i++) // i é o início do segmento corrente
    {
        sAux = 0;
        for (j = i; j < n; j++) // j é o final de tal segmento
        {
            sAux += v[j];
            if (sAux > *psMax)
            {
                *psMax = sAux;
                *pe = i;
                *pd = j;
            }
        }
    }
}
```

Invariantes e corretude:

- Observe que, na i-ésima iteração do laço externo
    - calculamos a soma de todos os segmentos que começam em i.
- De modo semelhante, na j-ésima iteração do laço interno
    - calculamos a soma dos segmentos que terminam em j.
- Os principais invariantes do laço externo são
    - v[*pe .. *pd] é um segmento de soma máxima de v com *pe < i.
    - *sMax = v[*e] + … + v[*d].
- O principal invariante do laço interno, que vale no início de cada iteração, é
    - sAux = v[i] + … + v[j - 1].

Eficiência de tempo:

- Número de operações no pior caso da ordem de n^2, i.e., O(n^2),
    - por conta dos dois laços aninhados,
        - cada um podendo ter tamanho da ordem de n.

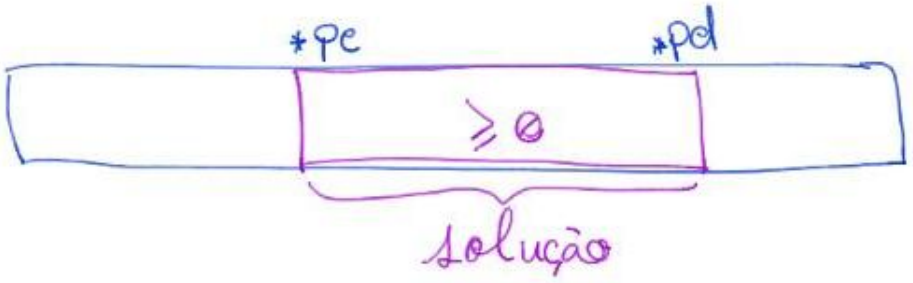
Eficiência de espaço:

- Memória auxiliar utilizada é constante em relação à entrada, i.e., O(1).

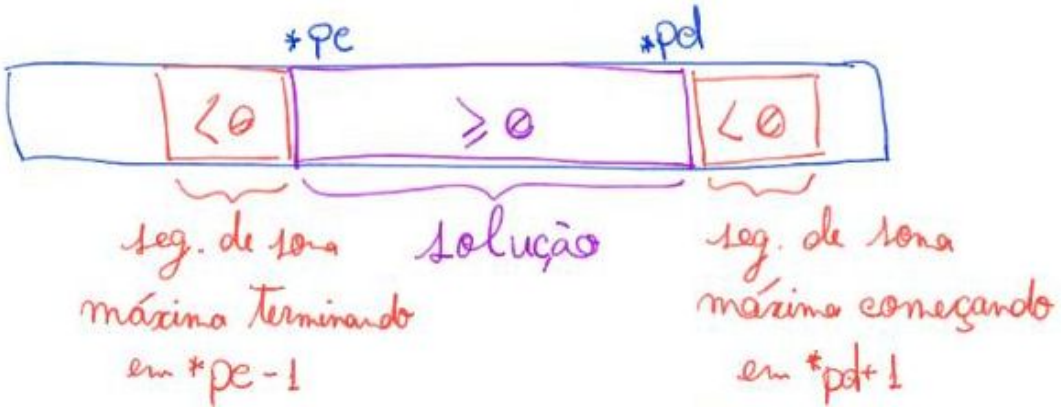
Será que conseguimos fazer melhor?

Observe que, considerando uma solução ótima qualquer no segmento v[*pe .. *pd],

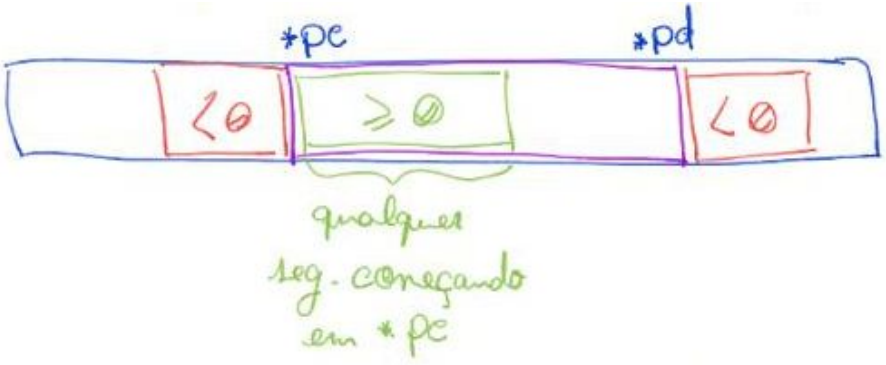
- podemos inferir algumas propriedades interessantes sobre esta.



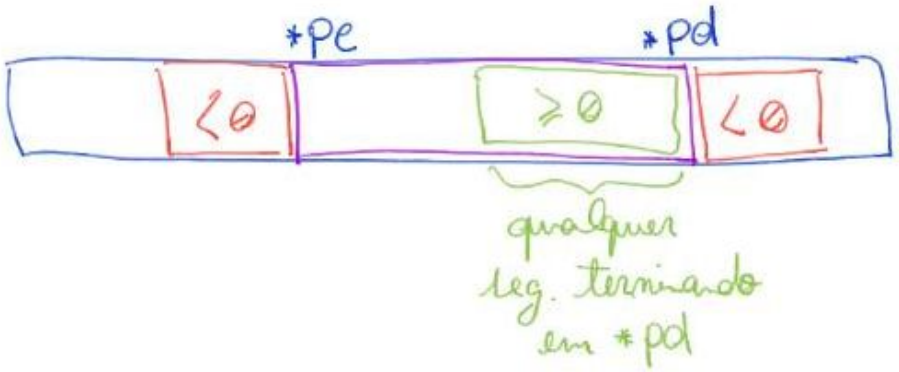
- Qualquer segmento que termina em *pe - 1,
  - i.e., da forma v[k .. *pe - 1], tem soma < 0.
    - Caso contrário, este teria sido adicionado à solução.
- O mesmo vale para qualquer segmento que começa em *pd + 1,
  - i.e., da forma v[*pd + 1 .. k].



- De modo complementar, qualquer segmento que começa em *pe
  - e está contido no subvetor da solução,
    - i.e., da forma v[*pe .. k] com k <= *pd, tem soma >= 0.
      - Caso contrário, este teria sido excluído da solução.

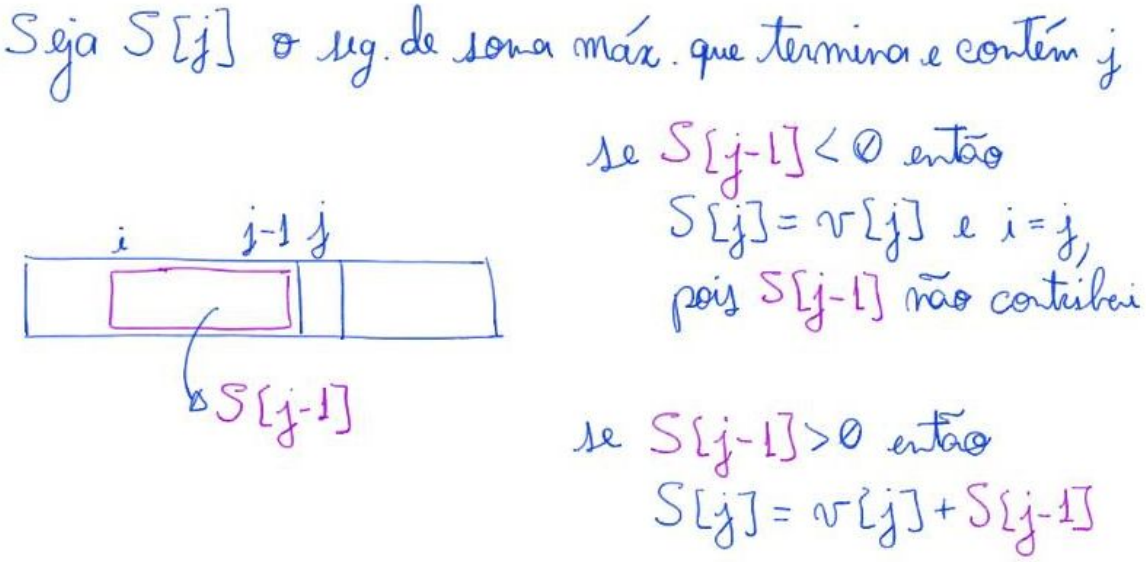


- O mesmo vale para qualquer segmento que termina em *pd
  - e está contido no subvetor da solução,
    - i.e., da forma v[k .. *pd] com k >= *pe.



Sabendo que uma solução ótima nunca começa com um prefixo de custo negativo,

- vamos abordar o problema usando um raciocínio recursivo (ou indutivo).
- Isto é, vamos pensar em como resolver nosso problema,
    - utilizando soluções para instâncias menores do problema.
- De fato, vamos pensar em instâncias de um problema relacionado,
    - em que o limite direito do intervalo é fixo.



- Seja S[j - 1] = v[i .. j - 1] um segmento de soma máxima
    - que termina e contém j - 1.
- Queremos calcular S[j], i.e., o segmento de soma máxima
    - que termina e contém j.
- Podemos usar S[j - 1] = v[i .. j - 1] para tanto?
    - Note que, nenhum outro segmento terminado em j - 1
        - pode contribuir tanto para S[j] quanto S[j - 1],
            - já que S[j - 1] é o máximo dos terminados em j - 1.
- Se S[j - 1] >= 0 então acrescentamos a ele v[j], i.e.,
    - S[j] = v[i .. j] é o segmento de soma máxima que termina e contém j.
- Caso contrário, todo segmento que termina e contém j - 1 tem soma < 0,
    - i.e., nenhum destes segmentos contribui para S[j].
- Portanto, S[j] = v[j] é o segmento de soma máxima que termina e contém j.

Essa ideia está por trás do seguinte algoritmo

```
// encontra o segmento de soma máxima em v[0 .. n - 1] e
// devolve seus limites em *pe, *pd e valor em *psMax
void segMax1(int v[], int n, int *pe, int *pd, int *psMax)
```

```
{
    int i, j, sAux;
    *psMax = 0;
    *pe = *pd = -1;
    // seja S[j - 1] o seg. soma máx. que termina e contém j - 1
    sAux = 0;                    // SAux guarda o valor de S[j - 1]
    for (i = j = 0; j < n; j++) // i é o início de S[j - 1]
    {
        if (sAux >= 0)
            sAux += v[j];
        else // sAux < 0
        {
            sAux = v[j];
            i = j;
        }
        if (sAux > *psMax)
        {
            *psMax = sAux;
            *pe = i;
            *pd = j;
        }
    }
}
```

Invariantes e corretude:

- Observe que, na j-ésima iteração do laço
    - v[*pe .. *pd] é um segmento de soma máxima de v[0 .. j - 1],
    - *psMax = v[*pe] + … + v[*pd],
    - v[i .. j - 1] é um segmento de soma máxima com término em j - 1,
    - sAux = v[i] + … + v[j - 1].

Eficiência de tempo:

- Número de operações no pior caso da ordem de n, i.e., O(n).

Eficiência de espaço:

- Memória auxiliar utilizada é constante em relação à entrada, i.e., O(1).

Uma versão levemente diferente, mas equivalente ao algoritmo anterior.

```
// encontra o segmento de soma máxima em v[0 .. n - 1] e
// devolve seus limites em *pe, *pd e valor em *psMax
void segMax0(int v[], int n, int *pe, int *pd, int *psMax)
{
    int i, j, sAux;
    *psMax = 0;
```

```
    *pe = *pd = -1;
    // seja S[j - 1] o seg. soma máx. que termina, mas pode não
conter j - 1
    sAux = 0;                   // SAux guarda o valor de S[j - 1]
    for (i = j = 0; j < n; j++) // i é o início de S[j - 1]
    {
        if (sAux + v[j] > 0)
            sAux += v[j];
        else // sAux + v[j] <= 0
        {
            sAux = 0;
            i = j + 1;
        }
        if (sAux > *psMax)
        {
            *psMax = sAux;
            *pe = i;
            *pd = j;
        }
    }
}
```

- Note que, este algoritmo usa uma variante da ideia recursiva
    - descrita anteriormente, na qual S[j - 1] = v[i .. j - 1]
        - é um segmento de soma máxima que termina em j - 1,
            - mas não necessariamente o contém,
                - i.e., o seguimento vazio é uma opção.