Aula 4

- Heaps
  - Filas de Prioridade
- Programação Dinâmica
  - Problema da Mochila
    (com e sem repetição)

Filas de Prioridade

- $Max(A)$: retorna a chave máxima da fila de prioridade
- $Extract(A)$: retorna a chave máxima da fila de prioridade e remove a chave da fila
- $Increase\,key(A, i, k)$: actualizar o valor da chave associado ao índice i para k
  Supondo que: $A[i] \leq k$
- $Insert\,key(A, k)$: introduz a chave k no heap

Filas de Prioridade

- Max(A): retorna a chave máxima da fila de prioridade

  Max(A)

- Extract(A): retorna a chave máxima da fila de prioridade e remove a chave da fila

  Extract(A)

Filas de Prioridade

- $Max(A)$: Retorna a chave máxima da fila de prioridade

```
Max(A)
    return A[1]
```

$O(1)$

- $Extract(A)$: retorna a chave máxima da fila de prioridade e remove a chave da fila

```
Extract(A)
    v := A[1];
    A[1] := A[A.size()];
    A.size --;
    MaxHeapify(A,1)
    Return v
```

$O(\log n)$

Filas de Prioridade

- Increase key (A, i, k) : actualizar o valor da chave associado ao índice i para k

  Supondo que: $\underline{A[i] \leq k}$

```
Increasekey (A, i, k)
   assert( A[i] ≤ k )
   if (i == 1) || (A[Parent(i)] ≥ k)
      A[i] := k
   else {
        A[i] := A[Parent(i)]
        Increasekey (A, Parent(i), k)
   }
```

$O(\log n)$

Filas de Prioridade

- InsertKey(A, k) : introduz a chave $\underline{k}$ no heap

```
InsertKey(A, k)
    A.size + 1;
    A[A.size] := -∞
    IncreaseKey(A, A.size, k)
```

$\underline{\underline{O(\lg n)}}$

Problema da Mochila NÃO Fraccionária

- Problema da mochila não fraccionária
  - Com replicação: dispomos de quantidades ilimitadas de cada item
  - Sem replicação: dispomos de uma quantidade limitada de cada item

- Problema da mochila não fraccionária com repetição

  Input:
  - $\vec{w}[1, \dots, n]$: vector de pesos
  - $\vec{v}[1, \dots, n]$: vector de valores

  Output:
  - k : valor que conseguimos transportar na mochila.

Problema da Mochila <u>NÃO</u> Fraccionária

- Problema da mochila não fraccionária
  - Com replicação: dispomos de quantidades ilimitadas de cada item
  - Sem replicação: dispomos de uma quantidade limitada de cada item

- Problema da mochila não fraccionária <u>com repetição</u>

<u>Input</u>:
- $\vec{w}[1, \dots, n]$: vector de pesos
- $\vec{v}[1, \dots, n]$: vector de valores

<u>Output</u>:
- K : valor que conseguimos transportar na mochila.

<u>Solução Recursiva</u>:

$$\underbrace{K(W)} = \max \left\{ K(W - w_k) + v_k \;\middle|\; \begin{array}{c} 1 \le k \le n, \\ w_k \le W \end{array} \right\}$$

valor q conseguimos transportar numa mochila com capacidade W

Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição

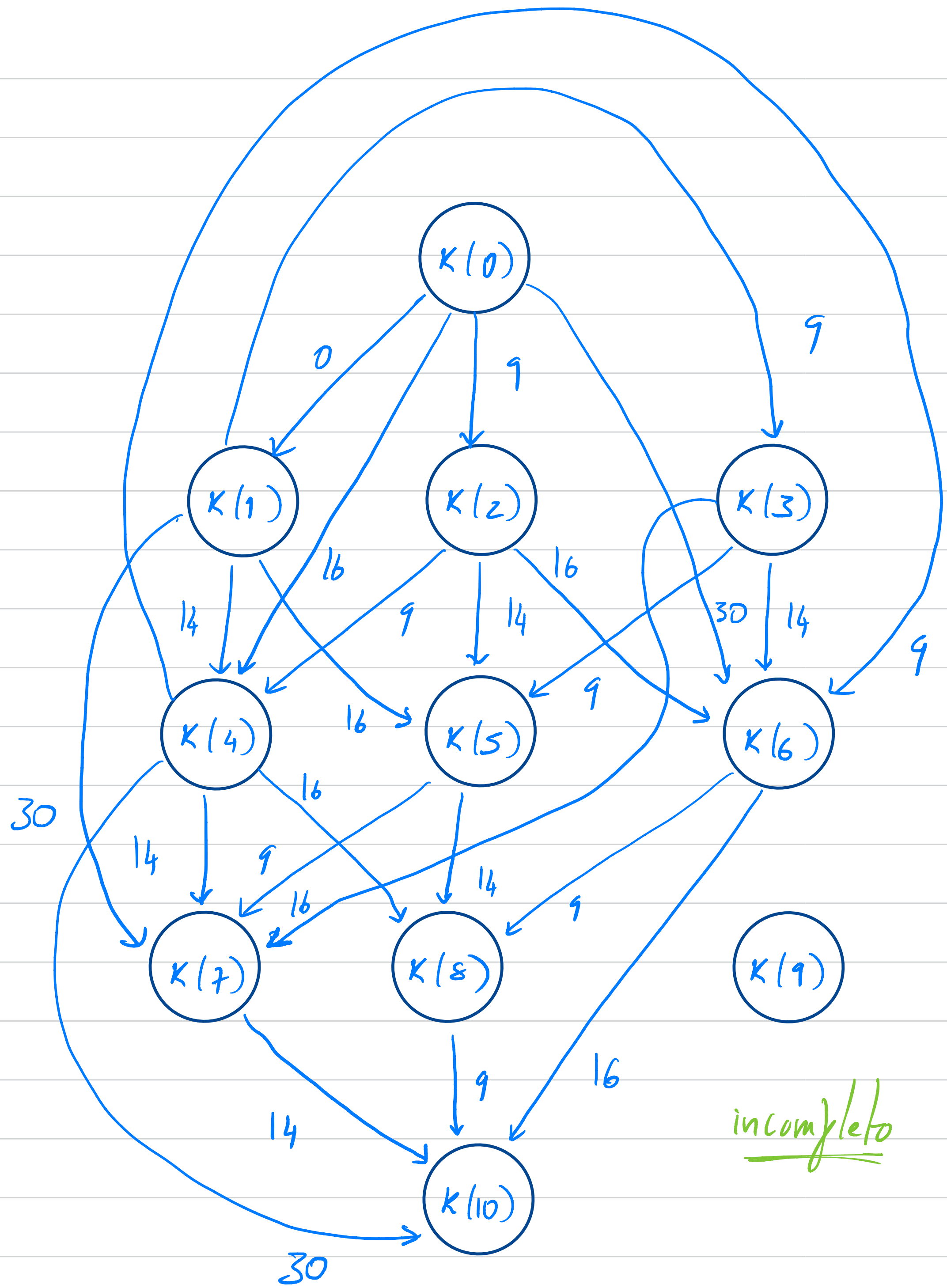
  - $K(W)$: maior valor que conseguimos transportar numa mochila com capacidade W

$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

Exemplo:

$W = 10$

Item	Peso	Valor
1	6	30 \$
2	3	14 \$
3	4	16 \$
4	2	9 \$



Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição

  - $K(W)$: maior valor que conseguimos transportar numa mochila com capacidade W

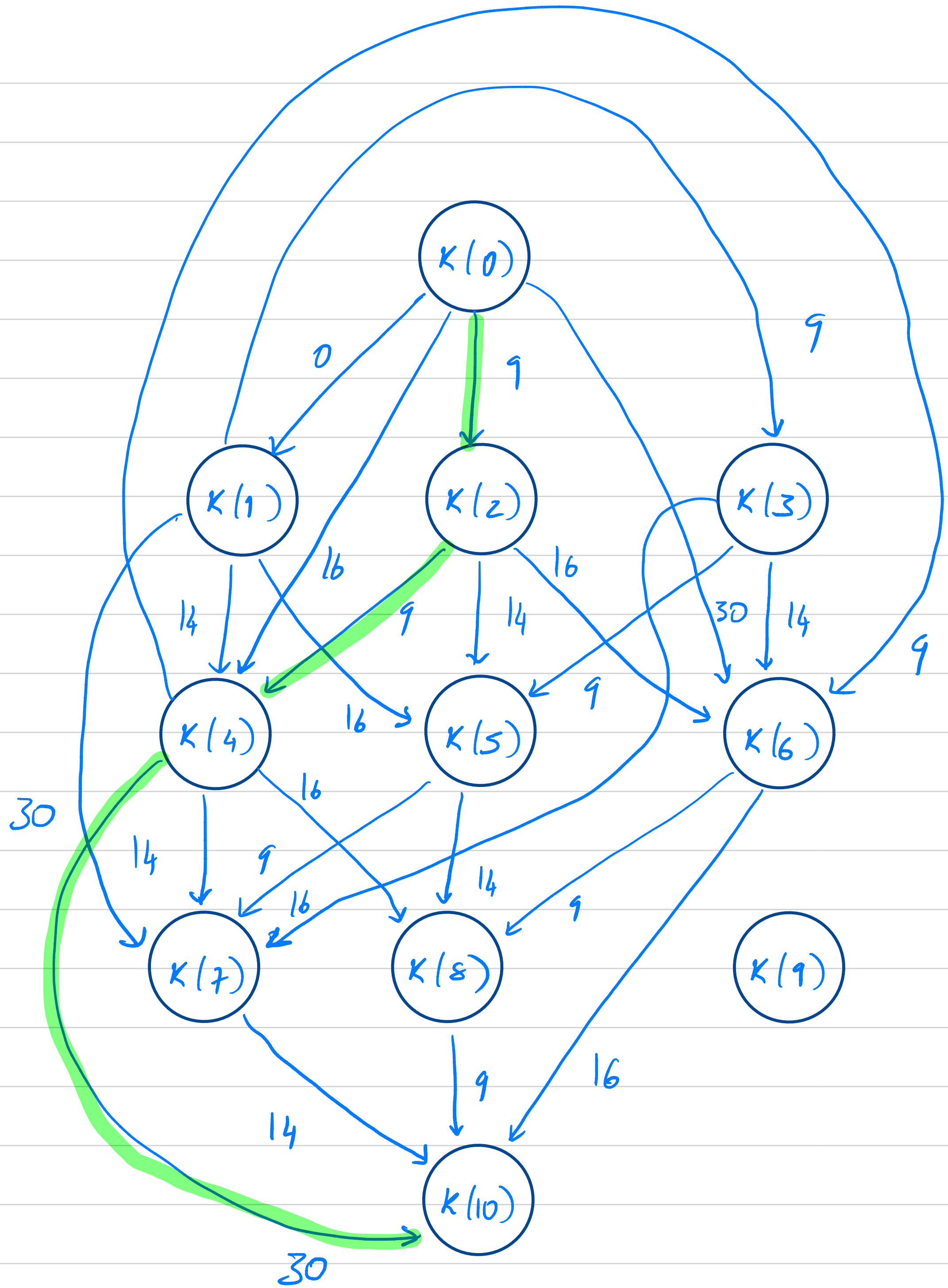
$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

Exemplo:

$W = 10$

Item	Peso	Valor
1	6	30 \$
2	3	14 \$
3	4	16 \$
4	2	9 \$

Caminho mais longo em DAG



Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição – Implementação naif

$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

```
knapsack (W, v⃗, w⃗, n)
    let k = 0
    for i = 1 to n
        if w⃗[i] ≤ W
            k := max(k, knapSack(W - w⃗[i], v⃗, w⃗, n))
    return k
```

Problema da Mochila Não Fraccionária

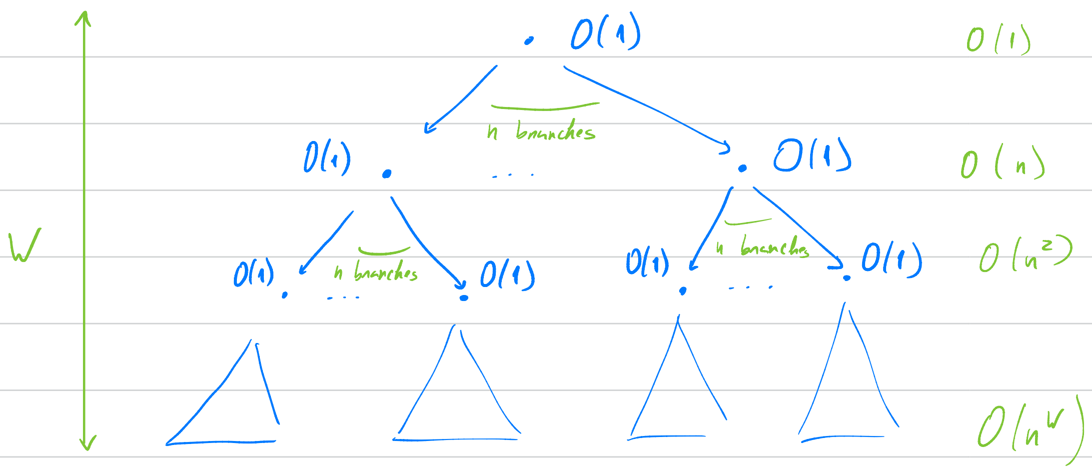
- Problema da Mochila com repetição – Implementação naif

$$K(W) = \max\{K(W - w_k) + v_k \mid w_k \leq W\}$$

```
knapsack (W, v⃗, w⃗, n)
  let k = 0
  for i = 1 to n
    if w⃗[i] ≤ W
      k := max(k, knapSack(W - w⃗[i], v⃗, w⃗, n))
  return k
```

Análise de Complexidade

$$T(W) = \begin{cases} n \times T(W-1) & \text{se } W > 0 \\ O(1) & \text{se } W = 0 \end{cases}$$



$$\underline{\underline{T(W) = O(n^W)}}$$

Problema da Mochila Não Fraccionária

- Problema da Mochila com repetição - Programação Dinâmica

$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

Knapsack ($\underline{W}$, $\vec{v}$, $\vec{w}$, n)

```
let k⃗[0..W] be a new vector initialized to 0
for w=1 to W
    for i:=1 to n
        if (w⃗[i] ≤ w)
            k⃗[w] = max ( k⃗[w], k⃗[w - w⃗[i]] + v⃗[i] )
```

Problema da Mochila Não Faccionária

- Problema da Mochila sem repetição – Programação Dinâmica

$$K(W) = \max \{ K(W - w_k) + v_k \mid w_k \leq W \}$$

Knapsack ($\underline{W}$, $\vec{v}$, $\vec{w}$, n)

let $\vec{k}[0..W]$ be a new vector
for $w = 1$ to $\underline{W}$
    for $i := 1$ to n
        if ($\vec{w}[i] \leq w$)
            $\vec{k}[w] = \max(\vec{k}[w], \vec{k}[w - \vec{w}[i]] + \vec{v}[i])$

Análise de Complexidade

$\Theta(n \cdot W)$

Problema da Mochila Não Fraccionária

Programação Dinâmica versus Memoization

```
Knapsack(W, v⃗, w⃗, n)
    let k⃗[0..W] be a new vector
    k⃗[0] := 0
    for w=1 to W
        for i:=1 to n
            if (w⃗[i] ≤ w)
                k⃗[w] = max(k⃗[w], k⃗[w - w⃗[i]] + v⃗[i])
```

Programação Dinâmica
- Construção Incremental da tabela de soluções

```
KnapSack'(W, k⃗, v⃗, w⃗, n)
    if (k⃗[W] ≠ Nil) return k⃗[W]
    let r := 0
    for i=1 to n
        if w⃗[i] ≤ W
            r := max(r, KnapSack'(W - w⃗[i], k⃗, v⃗, w⃗, n) + v⃗[i])
    k⃗[W] := r
    return r
```

Memoization
- A tabela é preenchida à medida q vamos precisando dos valores respectivos.

Problema da Mochila Não Fraccionária Sem Repetição

- Podemos usar cada elemento exactamente uma vez

$$K(\underline{W}, j) = \boxed{\cdots}$$

↳ Valor máximo q podemos transportar numa mochila c/ capacidade $\underline{W}$ usando unicamente elementos do conjunto $\{1, \dots, n\}$.

knapsackNoRep$(\underline{W}, \vec{v}, \vec{w}, n)$
  let $\vec{K}[\ ][\ ]$ be a $\underline{W} \times n$ array

  for $w = 0$ to $\underline{W}$
    $\vec{K}[w, 0] := 0$
  for $i = 0$ to n
    $\vec{K}[0, i] := 0$

} inicialização

Problema da Mochila Não Fraccionária Sem Repetição

- Podemos usar cada elemento exactamente uma vez

$$K(\underline{W}, j) = \begin{cases} \max\left(K(\underline{W}, j-1),\ K(\underline{W} - w_j, j-1) + v_j \right) & \text{se } \underline{W} \geq w_j \\ K(\underline{W} - w_j, j-1) & \underline{\text{c.c}} \end{cases}$$

knapsackNoRep$(\underline{W}, \vec{v}, \vec{w}, n)$
  let $\vec{K}[\,][\,]$ be a $\underline{W} \times n$ array

  for $w = 0$ to $\underline{W}$
    $\vec{K}[w, 0] := 0$
  for $i = 0$ to n
    $\vec{K}[0, i] := 0$

inicialização

Problema da Mochila Não Faccionária Sem Repetição

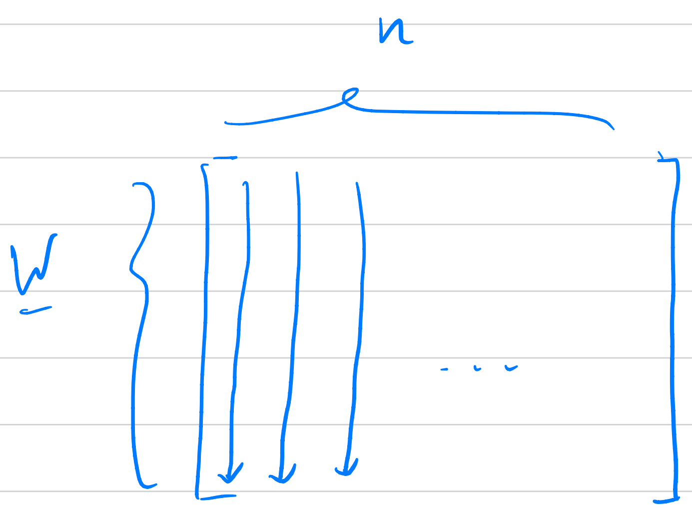
- Podemos usar cada elemento exactamente uma vez

```
knapsackNoRep(W, v⃗, w⃗, n)
  let K⃗[][] be a W×n array

  for w = 0 to W
    K⃗[w, 0] := 0
  for i = 0 to n
    K⃗[0, i] := 0

  for i = 1 to n
    for w = 1 to W
      K⃗[w, i] = max( K⃗[w, i-1], K⃗[w - w⃗[i], i-1] + v⃗[i] )

  return K⃗[W, n]
```



Problema da Mochila Não Faccionária Sem Repetição

- Podemos usar cada elemento exactamente uma vez

$$K(\underline{W}, j) = \max\left(K(\underline{W}, j-1),\ K(\underline{W} - w_j, j-1) + v_j \right)$$

```
knapsackNoRep(W, v⃗, w⃗, n)
  let K⃗[][] be a W x n array

  for w = 0 to W
    K⃗[w, 0] := 0
  for i = 0 to n
    K⃗[0, i] := 0

  for i = 1 to n
    for w = 1 to W
      K⃗[w, i] = max( K⃗[w, i-1], K⃗[w - w⃗[i], i-1] + v⃗[i] )

  return K⃗[W, n]
```

Análise de Complexidade

$$\Theta(n \cdot W)$$