Dynamické programování

Vojtěch Franc

Czech Technical University, Faculty of Electrical Engineering
Department of Cybernetics, Center for Machine Perception
121 35 Praha 2, Karlovo nám. 13, Czech Republic

xfrancv@cmp.felk.cvut.cz, http://cmp.felk.cvut.cz

Dynamické programování

- $F\colon \mathcal{X}^n \to \mathcal{R}$ ... kriteriální funkce přiřazuje reálné číslo každé posloupnosti $(x_1, \ldots, x_n) \in \mathcal{X} \times \mathcal{X} \times \cdots \times \mathcal{X}$.
- $\mathcal{X} = \{1, \ldots, K\}$ ... konečná množina s K prvky. Tudíž počet všech posloupností $(x_1, \ldots, x_n) \in \mathcal{X}^n$ je roven K^n.
- Naším cílem je vyřešit minimalizační úlohu

$$F^* = \min_{(x_1,\ldots,x_n)\in\mathcal{X}^n} F(x_1, \ldots, x_n)$$

- Úloha je polynomiálně řešitelná pomocí dynamického programování pokud

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{n-1} f_i(x_i, x_{i+1})$$

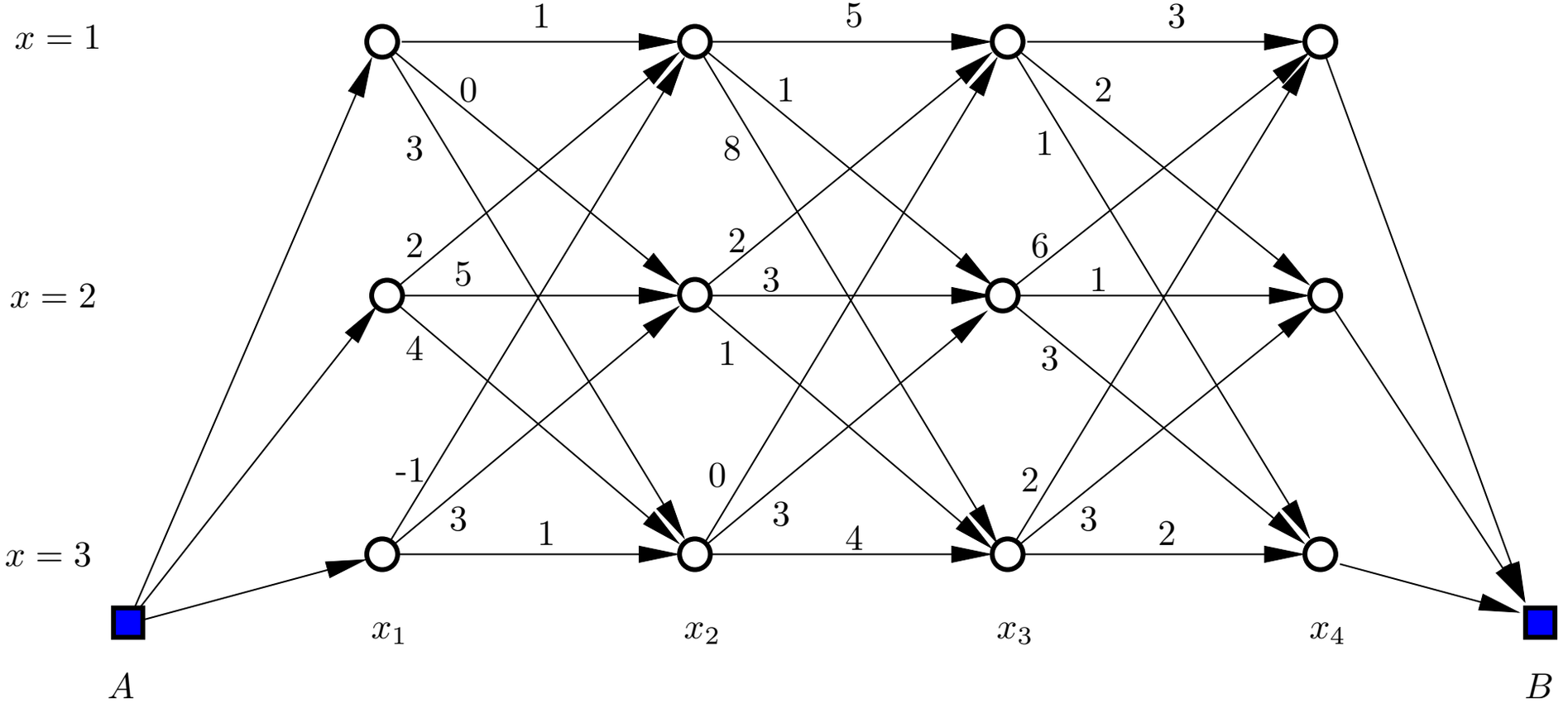
  kde $f_i\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $j = 1, \ldots, n-1$ je množina $n-1$ funkcí arity 2.
- Nekonvexní problém, protože $\mathcal{X}^n$ není konvexní množina. Úloha je instancí celočíselného programování (integer programing).

Dynamické programování

- Příklad: $\mathcal{X} = \{1, 2, 3\}$, $n = 4$

$$F^* = \min_{(x_1,x_2,x_3,x_4)\in\mathcal{X}^4} \left[f_1(x_1, x_2) + f_2(x_2, x_3) + f_3(x_3, x_4)\right]$$

což je ekvivalentní hledání nejkratší cesty v orientovaném acyklickém grafu na jehož hranách jsou hodnoty funkcí $f_1(x_1, x_2), \ldots, f_{n-1}(x_{n-1}, x_n)$



Dynamické programování

$$
\begin{aligned}
F^* &= \min_{x_1,\ldots,x_n} \left[\sum_{i=1}^{n-2} f_i(x_i, x_{i+1}) + f_{n-1}(x_{n-1}, x_n) \right] \\
&= \min_x F_n(x) \\
F_n(x) &= \min_{x_1,\ldots,x_{n-1}} \left[\sum_{i=1}^{n-2} f_i(x_i, x_{i+1}) + f_{n-1}(x_{n-1}, x) \right] \\
&= \min_{x'} \left[F_{n-1}(x') + f_{n-1}(x', x) \right] \\
F_{n-1}(x) &= \min_{x_1,\ldots,x_{n-2}} \left[\sum_{i=1}^{n-3} f_i(x_i, x_{i+1}) + f_{n-2}(x_{n-2}, x) \right] \\
&= \min_{x'} \left[F_{n-2}(x') + f_{n-2}(x', x) \right] \\
&\vdots \\
F_2(x) &= \min_{x'} f_1(x', x)
\end{aligned}
$$

Dynamické programování

Algoritmus dynamické programování

Inicializace: $F_1(x) = 0,\ \forall x \in \mathcal{X}$

for $i = 2, \ldots, n$ **do**

$$F_i(x) = \min_{x' \in \mathcal{X}} \left[F_{i-1}(x') + f_{i-1}(x', x) \right] \qquad x \in \mathcal{X}$$

end for

$F^* = \min_{x \in \mathcal{X}} F_n(x)$

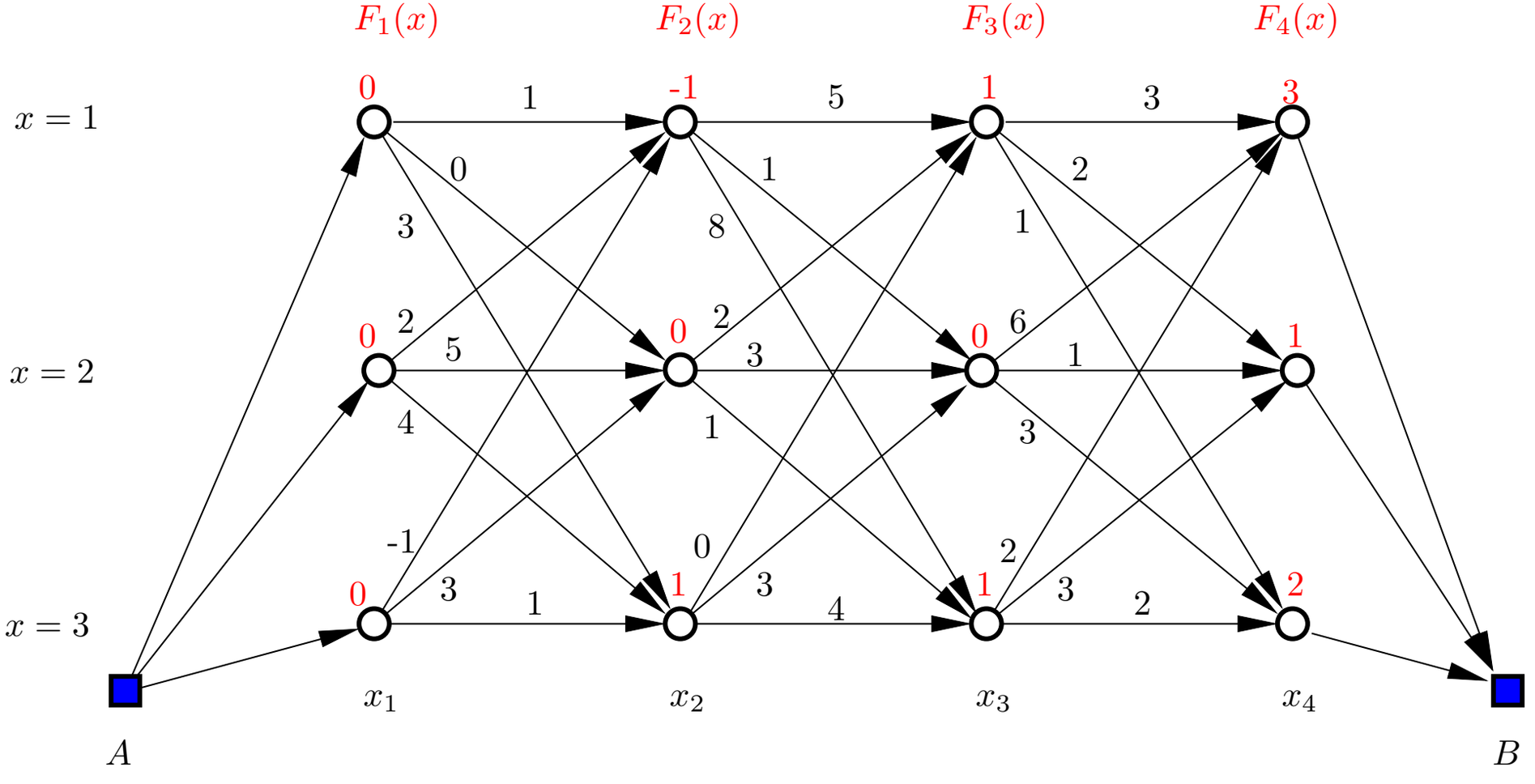
- Algoritmus nalezne globální minimum $F^* = \min_{(x_1,\ldots,x_n) \in \mathcal{X}^n} \sum_{i=1}^{n-1} f_i(x_i, x_{i+1})$.
- Složitost algoritmu je $\mathcal{O}(nK^2)$, kde n je počet proměnných a K je počet možných hodnot jedné proměnné.

Dynamické programování

- Příklad: $\mathcal{X} = \{1, 2, 3\}$, $n = 4$

$$F^* = \min_{(x_1,x_2,x_3,x_4)\in\mathcal{X}^4} \left[f_1(x_1, x_2) + f_2(x_2, x_3) + f_3(x_3, x_4)\right] = 1$$

$F_i(x)$ je cena nejkratší cesty z uzlu A do uzlu (i, x).



Dynamické programování

- Co když chceme znát optimální hodnoty proměnných, tj.

$$(x_1^*, \ldots, x_n^*) = \operatorname*{argmin}_{(x_1,\ldots,x_n)\in\mathcal{X}^n} F(x_1, \ldots, x_n)$$

Algoritmus dynamické programování

Inicializace: $F_1(x) = 0, \forall x \in \mathcal{X}$

for $i = 2, \ldots, n$ **do**

$$F_i(x) = \min_{x'\in\mathcal{X}} \left[F_{i-1}(x') + f_{i-1}(x', x)\right] \qquad x \in \mathcal{X}$$

$$ind_i(x) = \operatorname*{argmin}_{x'\in\mathcal{X}} \left[F_{i-1}(x') + f_{i-1}(x', x)\right] \qquad x \in \mathcal{X}$$

end for

$F^* = \min_{x\in\mathcal{X}} F_n(x)$

$x_n^* = \operatorname{argmin}_{x\in\mathcal{X}} F_n(x)$

for i=n,. . . ,2 **do**

$x_{i-1}^* = ind_i(x_i^*)$

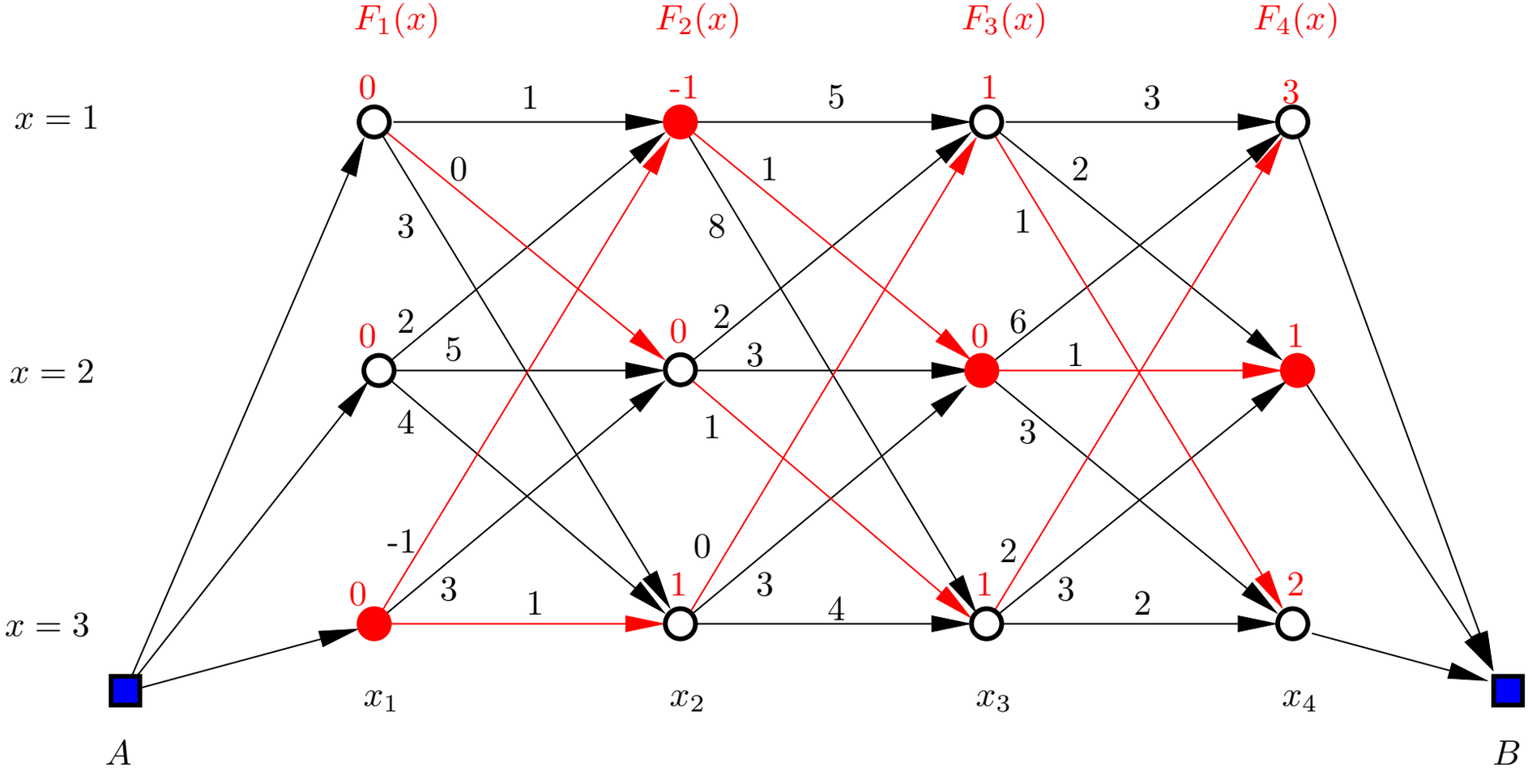
end for

Dynamické programování

- Příklad: $\mathcal{X} = \{1, 2, 3\}$, $n = 4$

$$(x_1^*, \ldots, x_4^*) = \min_{(x_1,x_2,x_3,x_4)\in\mathcal{X}^4} \left[f_1(x_1, x_2) + f_2(x_2, x_3) + f_3(x_3, x_4)\right] = (3, 1, 2, 2)$$

$(i-1, ind_i(x))$ je uzel, kterým prochází nejkratší cesta z uzlu A do uzlu (i, x).



Dynamické programování

- Dynamické programování poprvé použito Richardem Bellmanem pro řešení problémů z teorii řízení (1953). Později nalezlo aplikace v mnoha oborech.
- Dynamické programování je optimalizační strategie pro řešení složitých problémů na základě jejich postupné dekompozice na množinu jednodušších problémů.
- Podproblémy se řeší od nejjednodušších přes složitější až k původnímu problému.
- Řešení jednodušších problému se používá k vyřešení složitějších. Vzorec pro výpočet složitějšího problému z jednoduších se názývá bellmanova rovnice.
- Složitější problém je typicky jen o málo složitější než podmnožina jednodušších podproblémů, jejichž řešení je potřeba k jeho vyřešení.
- Dynamické programování implicitně definuje orientovaný acyklický graf jehož uzly jsou jednotlivé podproblémy a hrany určují pořadí řešení těchto podproblému.

Levensteinova vzdálenost

- Cílem je spočítat vzdálenost mezi dvěma řetezci $x \in A^n$ a $y \in \mathcal{A}^m$ nad abecedou $\mathcal{A}$.
- Editovací operace:

Operace	Ekvivalentní operace (klávesa) v editoru
DELETE	DEL
INSERT	(Insert mode) A ··· Z
SUBSTITUTE	(Overwrite mode) A ··· Z

Operace jdi na další znak → se nepočítá mezi editovacích operace, ale lze použít.

- Levenstainova vzdálenost $d(x, y)$ je minimální počet editovacích operací, které převedou řetězec x na řetězec y
- Levenstainova vzdálenost je metrika:
  - $d(x, y) \geq 0$ (nezápornost)
  - $d(x, y) = 0 \iff x = y$ (0 pouze když x a jsou shodné y)
  - $d(x, y) = d(y, x)$ (symetrická)
  - $d(x, z) = d(x, y) + d(y, z)$ (trojúhelníková nerovnost)

Levensteinova vzdálenost

Příklad: $x =$"DWARF", $y =$"WORM"

"DWARF" $\Rightarrow$ "WORM"

řetězec	operace	penalta
D WARF	DELETE	1
W ARF	jdi na další znak	0
W A RF	SUBST A$\Rightarrow$ O	1
WO R F	jdi na další znak	0
WOR F	SUBST F$\Rightarrow$ M	1
WORM	počet edit. operací je 3	

"WORM" $\Rightarrow$ "DWARF"

řetězec	operace	penalta
W ORM	INSERT D	1
D W ORM	jdi na další znak	0
DW O RM	SUBST O$\Rightarrow$ A	1
DWA R M	jdi na další znak	0
DWAR M	SUBST M$\Rightarrow$ F	1
DWARF	počet edit. operací je 3	

Levensteinova vzdálenost

- Abychom mohli použít dynamické programování, musíme problém převést na řešení o něco jednodušších problémů.
- Nechť $F(i, j)$ je Levensteinova vzdálenost mezi $x[1 : i]$ a $y[1 : j]$, tj. $d(x, y) = F(n, m)$.
- Levensteinova vzdálenost $F(i, j)$ může být rovna jen jednomu ze čtyř čísel

  $F(i-1, j)+1$ pokud poslední operace je DELETE

  $F(i, j-1)+1$ pokud poslední operace je INSERT $y[j]$

  $F(i-1, j-1)+1$ pokud poslední operace je SUBSTITUTE $x[i] \Rightarrow y[i]$

  $F(i-1, j-1)$ pokud poslední operace je přejdi na další znak.

  a tudíž musí platit

$$F(i,j) = \min\left\{F(i-1,j)+1, F(i,j-1)+1, F(i-1,j-1)+\mathrm{diff}(x[i], y[j])\right\}$$

Levensteinova vzdálenost

- Příklad 1: $x =$"DWARF", $y =$"WORM"

vzdálenost	mezi řetězci	zbývá vyřešit
$F(4,4) = 3$	"DWAR" a "WORM"	WORM F
$F(5,3) = 3$	"DWARF" a "WOR"	WOR
$F(4,3) = 2$	"DWAR" a "WOR"	WOR F

$$d(x,y) = F(5,4) = \min\left\{3+1, 3+1, 2+\mathrm{diff}(''F'','' M'')\right\} = 3$$

- Příklad 2: $x =$"DWAR", $y =$"WOR"

vzdálenost	mezi řetězci	zbývá vyřešit
$F(3,3) = 3$	"DWA" a "WOR"	WOR R
$F(4,2) = 3$	"DWAR" a "WO"	WO
$F(3,2) = 2$	"DWA" a "WO"	WO R

$$d(x,y) = F(4,3) = \min\left\{3+1, 3+1, 2+\mathrm{diff}(''R'','' R'')\right\} = 2$$

Levensteinova vzdálenost

Algoritmus s výpočetní složitostí $\mathcal{O}(mn)$

Vstup: řetězce $x[1:n]$ a $y[1:m]$
for $i = 0, \ldots, n$ **do**
    $F(i, 0) = i$
end for
for $j = 0, \ldots, m$ **do**
    $F(0, j) = j$
end for
for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, n$ **do**

$$F(i,j) = \min \left\{ F(i-1,j)+1, F(i,j-1)+1, F(i-1,j-1)+\mathrm{diff}(x[i], y[j]) \right\}$$

    end for
end for
$d(x, y) = F(n, m)$

Levensteinova vzdálenost

Příklad: $x =$"DWARF", $y =$"WORM"

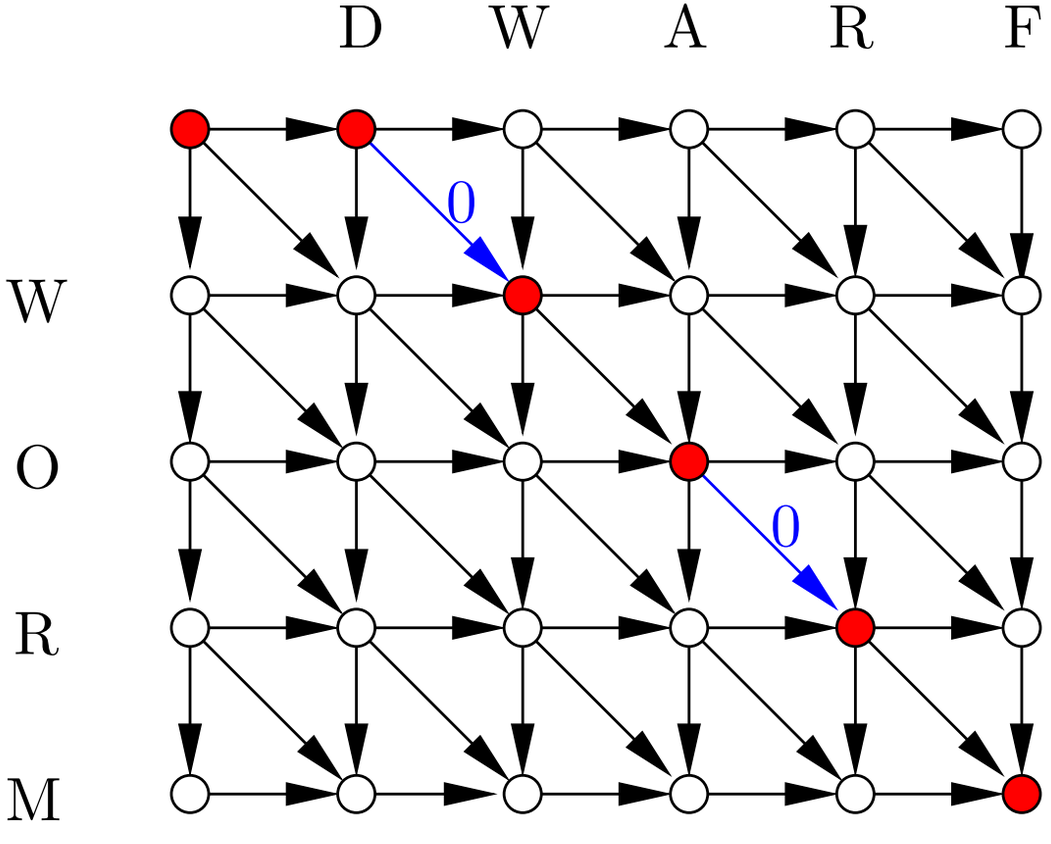
Tabulka vzdáleností $F(i, j)$

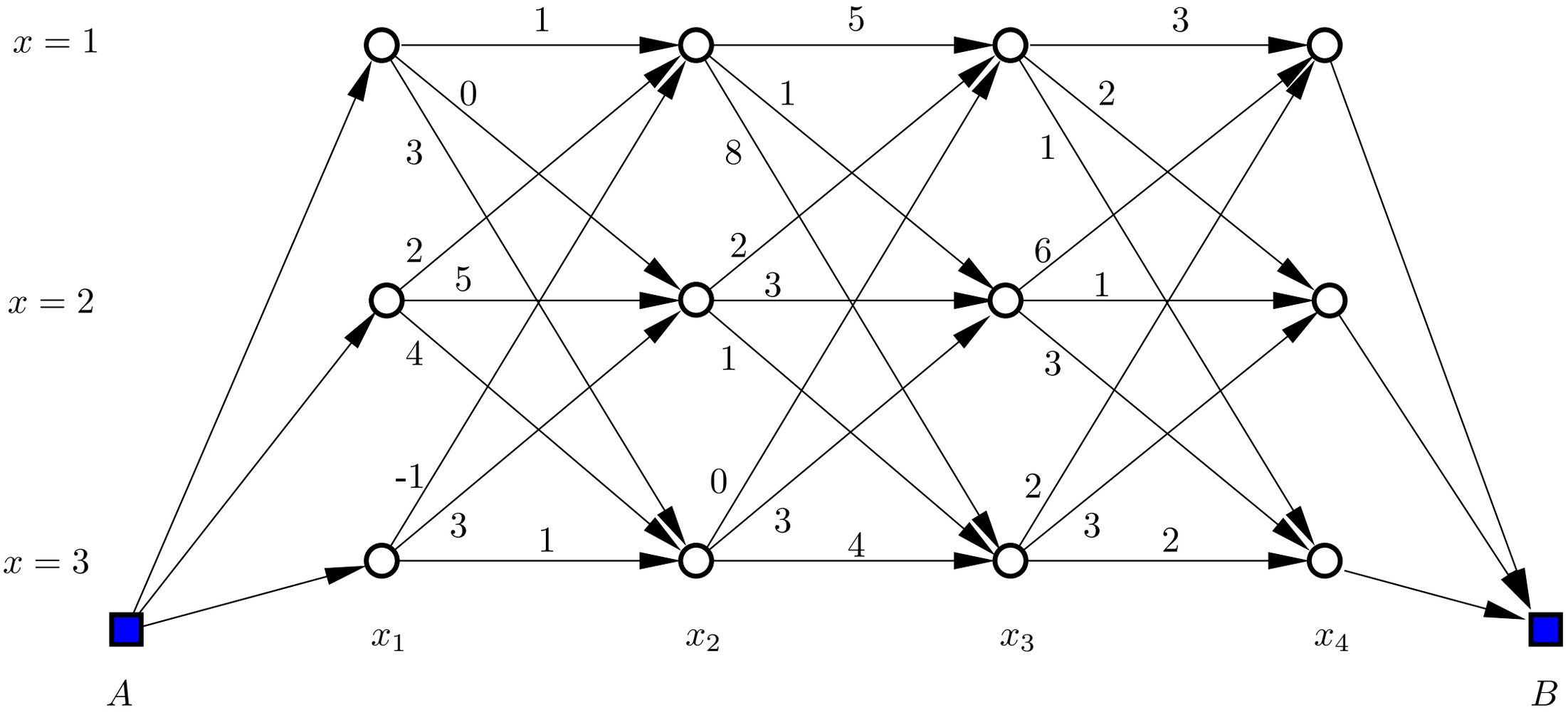
		D	W	A	R	F
	0	1	2	3	4	5
W	1	1	1	2	3	4
O	2	2	2	2	3	4
R	3	3	3	3	2	3
M	4	4	4	4	3	3

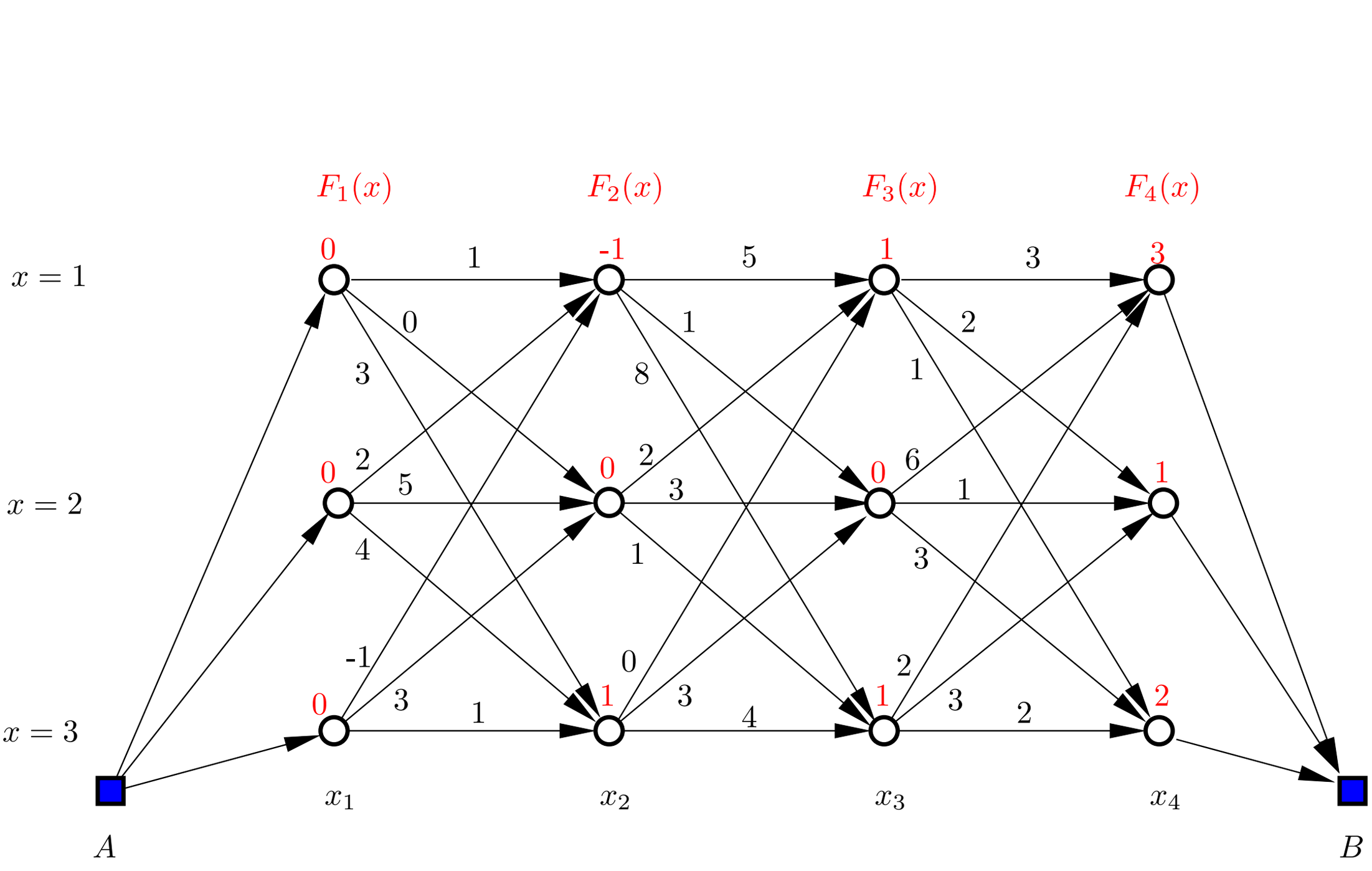
Levensteinova vzdálenost

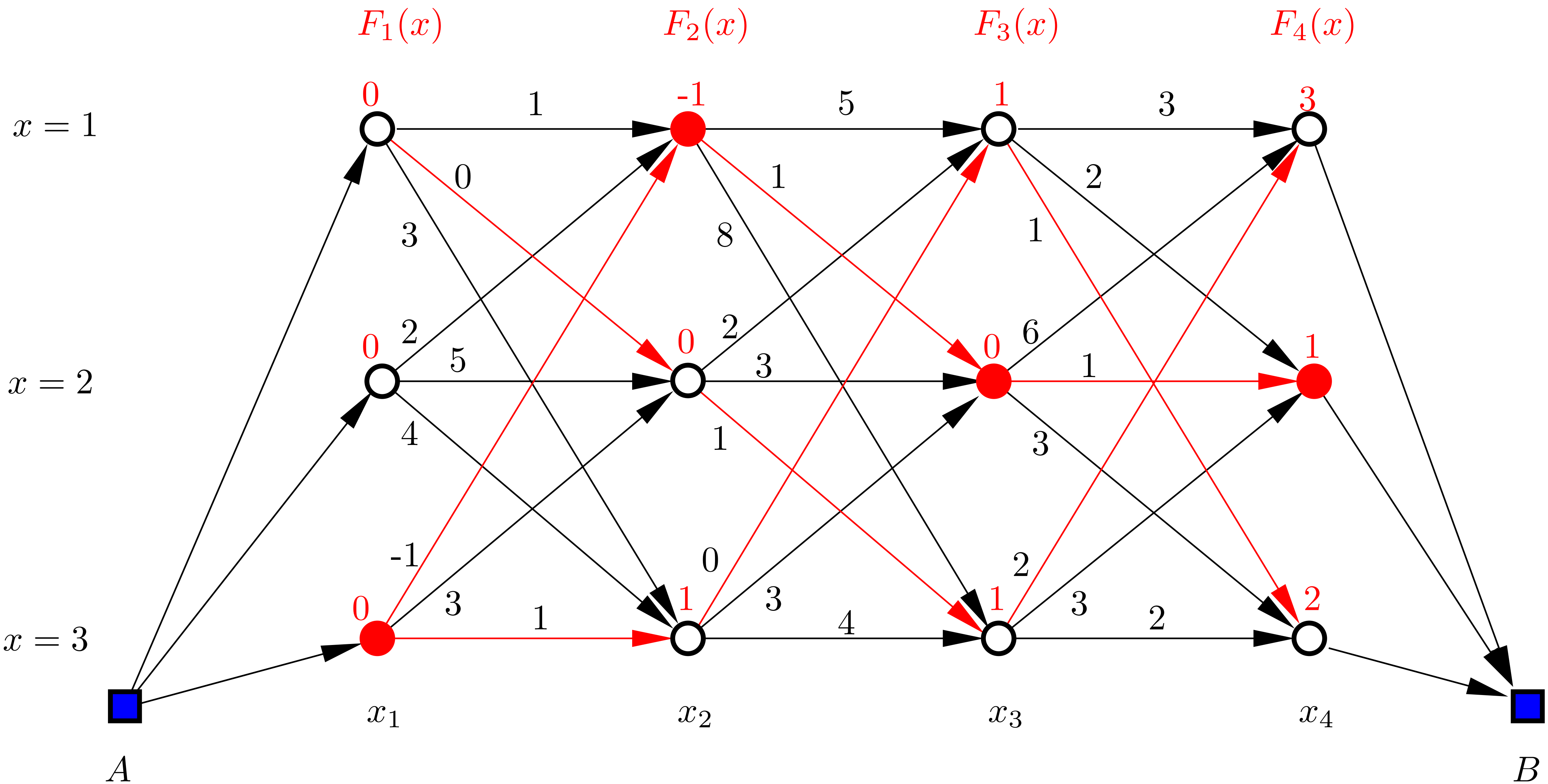
Výpočet Levensteinovy vzdálenosti je ekvivalentní hledání nejkratší cesty v orientovaném acyklickém grafu:

- Horizontální hrana odpovídá operaci DELETE (penalta 1)
- Vertikální hrana odpovídá operaci INSERT (penalta 1)
- Pokud $\mathrm{diff}(x[i], y[j]) = 1$ odpovídá diagonální hrana operaci SUBSTITUTE (penalta 1)
- Pokud $\mathrm{diff}(x[i], y[j]) = 0$ odpovídá diagonální hrana přechodu na další znak (penalta 0)

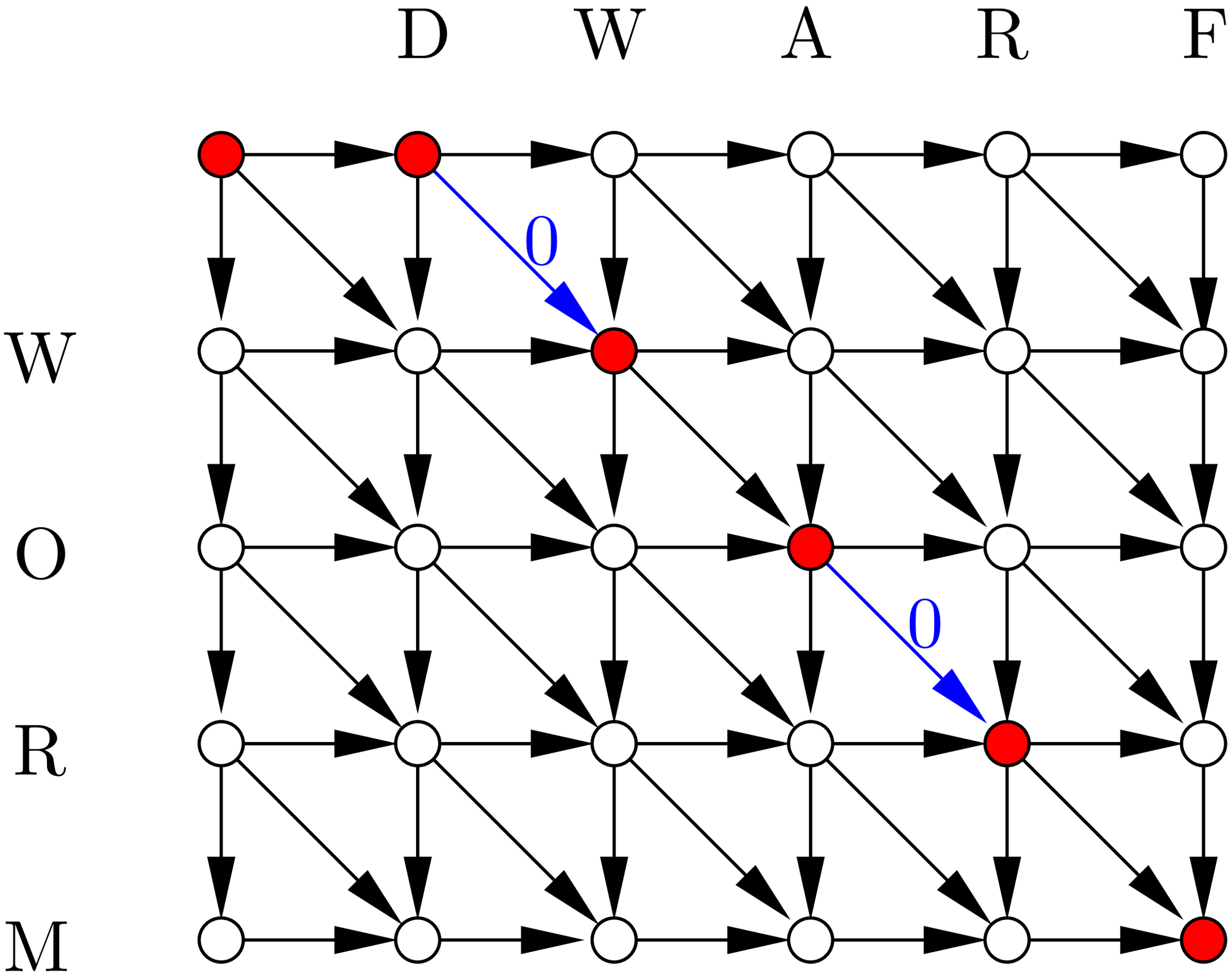



$x = 1$
$x = 2$
$x = 3$
1
5
3
0
1
2
3
8
1
2
5
3
1
4
1
3
-1
0
2
3
3
1
4
3
2
x_1
x_2
x_3
x_4
A
B


$F_1(x)$
$F_2(x)$
$F_3(x)$
$F_4(x)$
$x = 1$
$x = 2$
$x = 3$
0
-1
1
3
0
0
1
0
1
2
0
0
1
0
1
2
x_1
x_2
x_3
x_4
A
B



$F_1(x)$
$F_2(x)$
$F_3(x)$
$F_4(x)$
$x = 1$
$x = 2$
$x = 3$
x_1
x_2
x_3
x_4
A
B
0
-1
1
3
0
0
0
1
0
1
1
2
1
5
3
0
3
1
8
2
1
2
3
5
2
3
1
6
4
1
3
-1
0
2
3
3
3
1
4
2


D
W
A
R
F
W
O
R
M
0
0