Welcome To BigBlueButton

BigBlueButton is an open source web conferencing system designed for online learning

CHAT

Send public and private messages.

WEBCAMS

Hold visual meetings.

AUDIO

Communicate using high quality audio.

EMOJIS

Express yourself.

BREAKOUT ROOMS

Group users into breakout rooms for team collaboration.

POLLING

Poll your users anytime.

SCREEN SHARING

Share your screen.

MULTI-USER WHITEBOARD

Draw together.

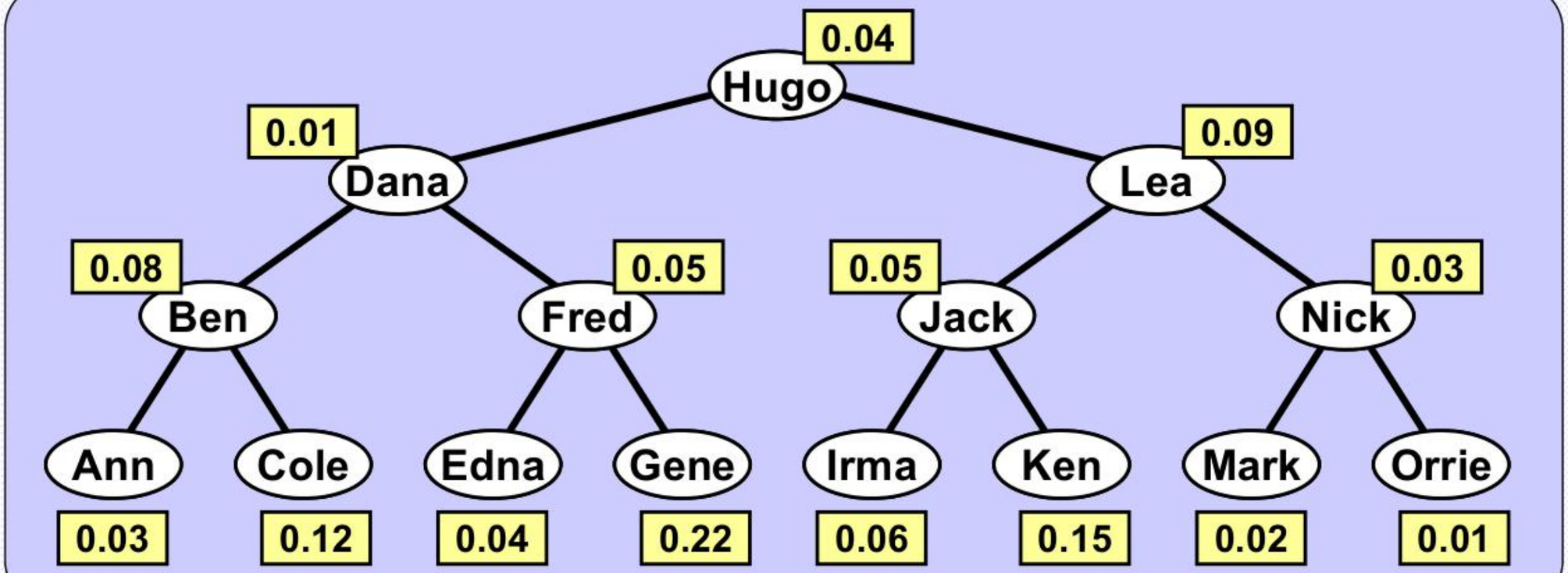
For more information visit bigbluebutton.org →

Dynamické programování

Optimální binární vyhledávací strom

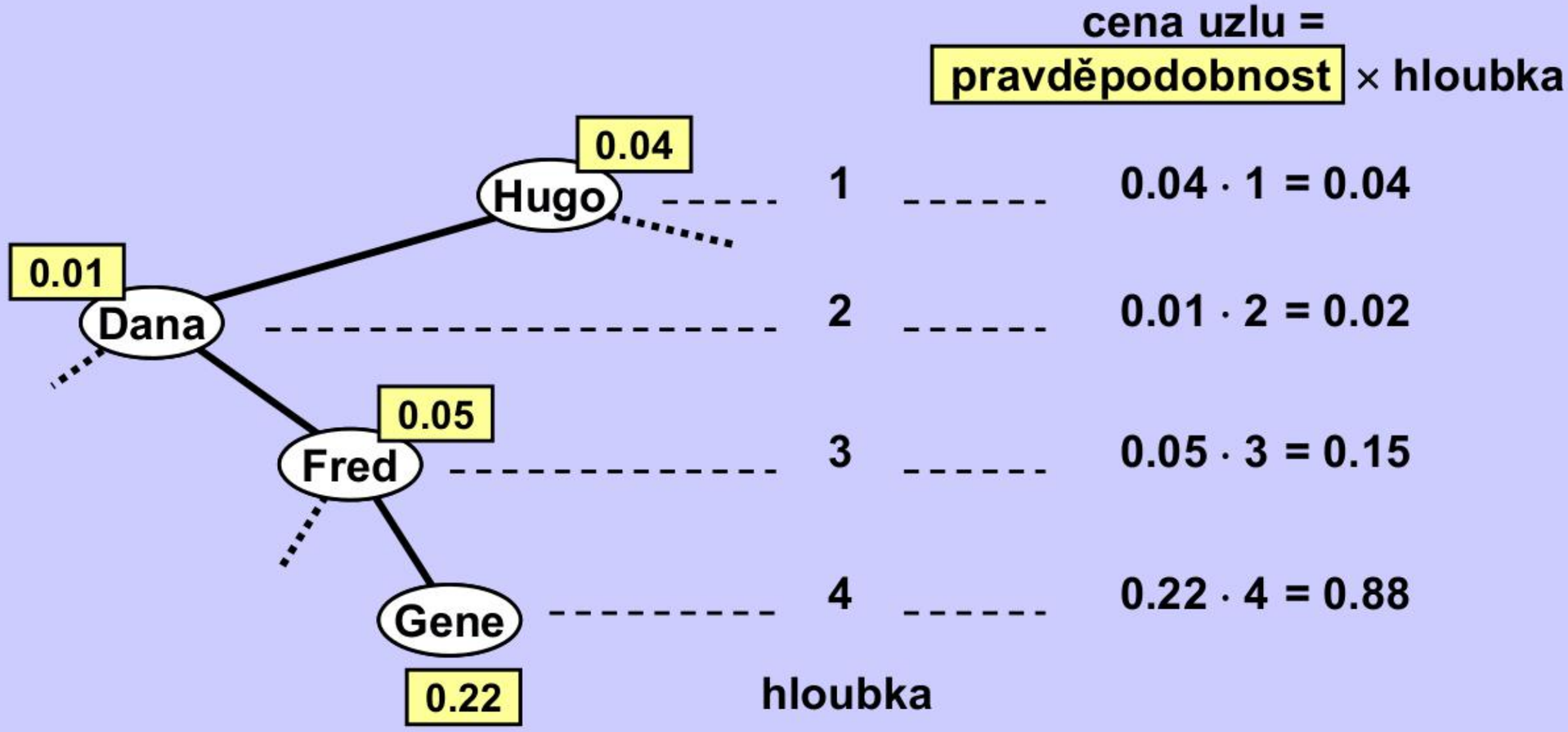
Optimální binární vyhledávací strom

Vyvážený, ale ne optimální



Optimální binární vyhledávací strom

Cena jednotlivých uzlů v BVS



cena uzlu = průměrný počet testů na nalezení uzlu při jednom dotazu (Find)

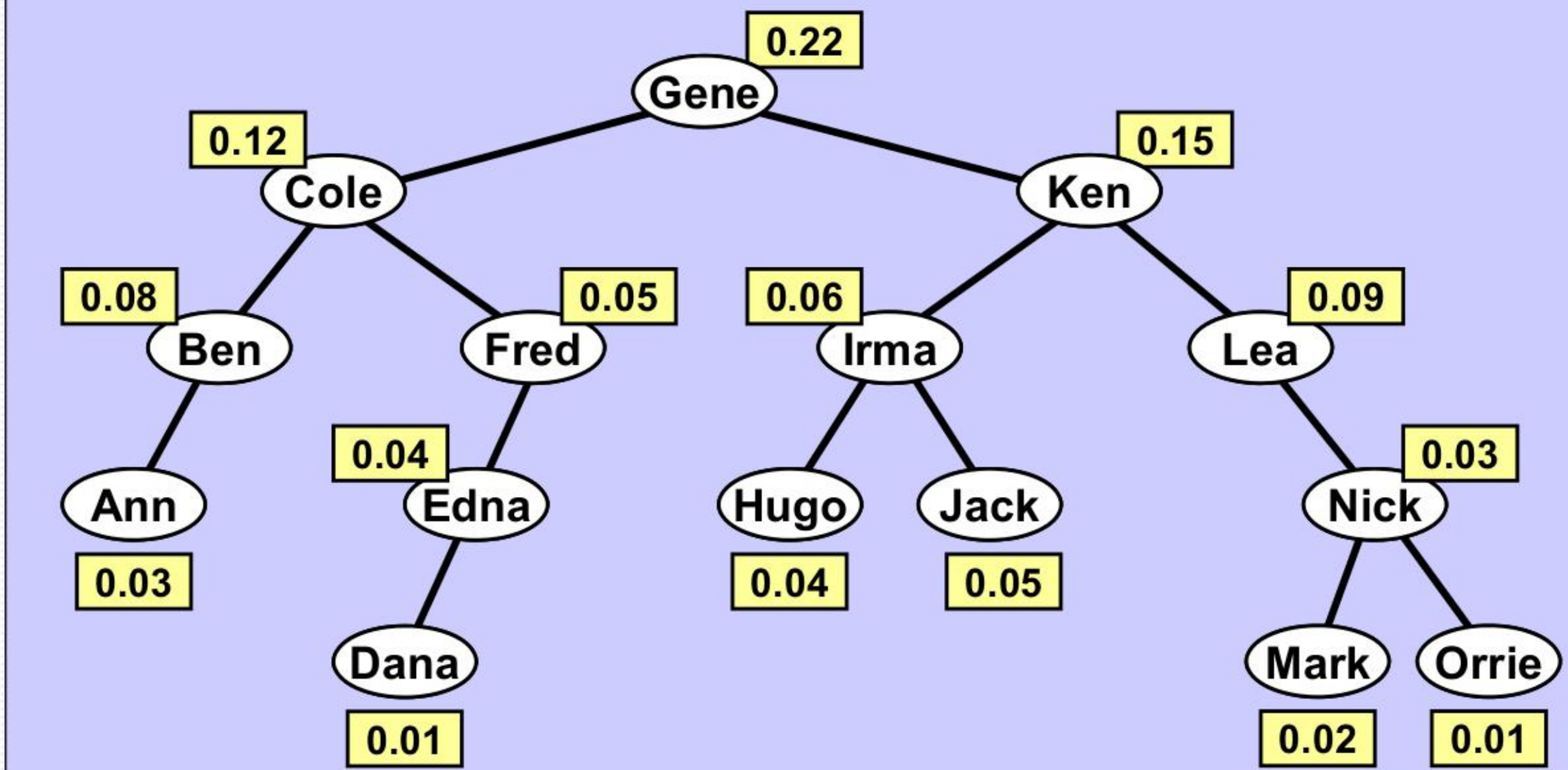
Cena vyváženého stromu

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	4	0.12 · 4 = 0.48
Dana	0.01	2	0.01 · 2 = 0.02
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	4	0.22 · 4 = 0.88
Hugo	0.04	1	0.04 · 1 = 0.04
Irma	0.06	4	0.06 · 4 = 0.24
Jack	0.05	3	0.05 · 3 = 0.15
Ken	0.15	4	0.15 · 4 = 0.60
Lea	0.09	2	0.09 · 2 = 0.18
Mark	0.02	4	0.02 · 4 = 0.08
Nick	0.03	3	0.03 · 3 = 0.09
Orrie	0.01	4	0.01 · 4 = 0.04
			Cena celkem: 3.47

Cena celkem = prům. poč. testů na jednu operaci Find.

Optimální BVS

Struktura optimálního BVS s danými pravděpodobnostmi

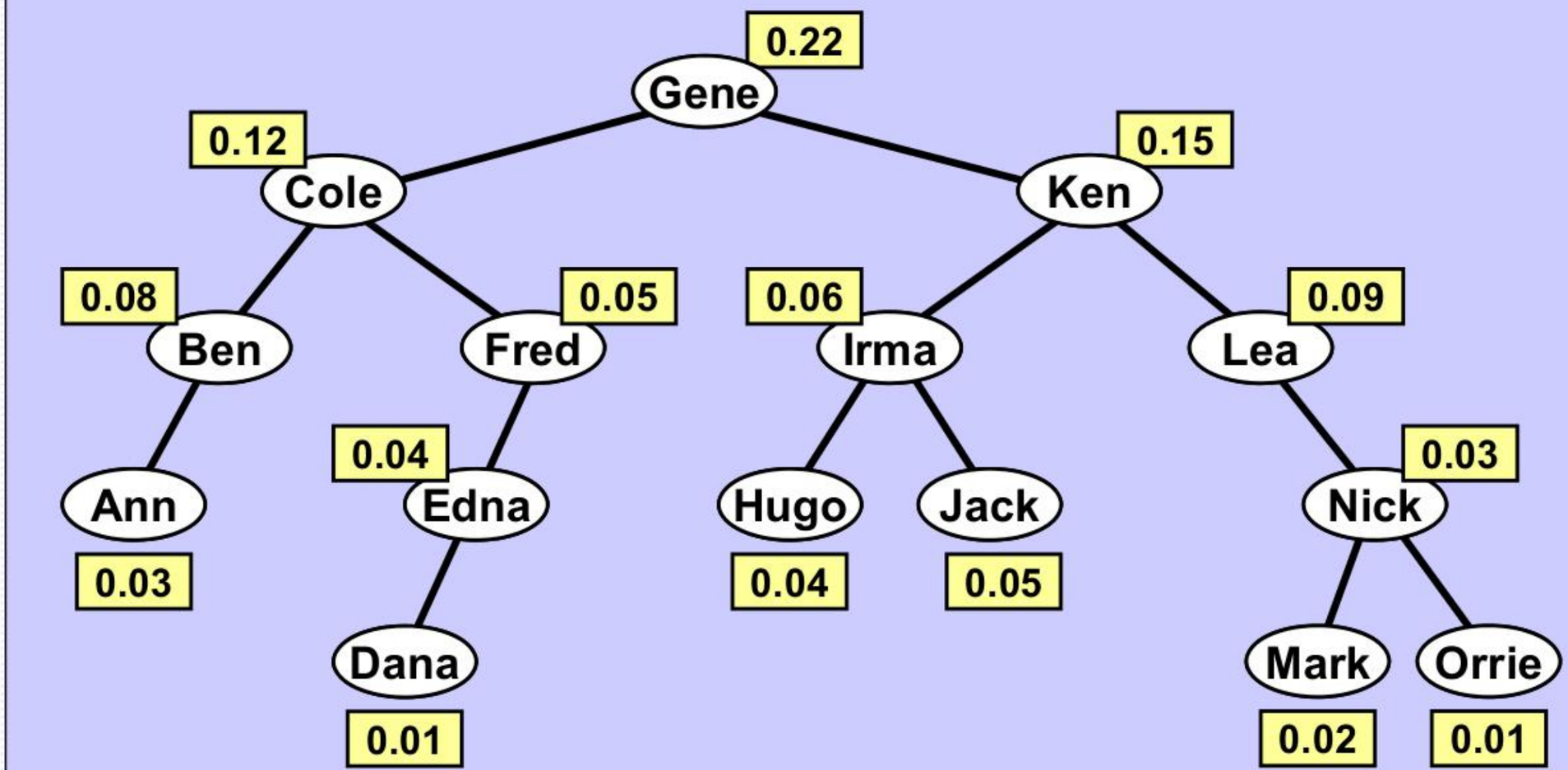


Cena optimálního BVS

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	2	0.12 · 2 = 0.24
Dana	0.01	5	0.01 · 5 = 0.05
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	1	0.22 · 1 = 0.22
Hugo	0.04	4	0.04 · 4 = 0.16
Irma	0.06	3	0.06 · 3 = 0.18
Jack	0.05	4	0.05 · 4 = 0.20
Ken	0.15	2	0.15 · 2 = 0.30
Lea	0.09	3	0.09 · 3 = 0.27
Mark	0.02	5	0.02 · 5 = 0.10
Nick	0.03	4	0.03 · 4 = 0.12
Orrie	0.01	5	0.01 · 5 = 0.05
			Cena celkem 2.56
Zrychlení 3.47 : 2.56 = 1 : 0.74			

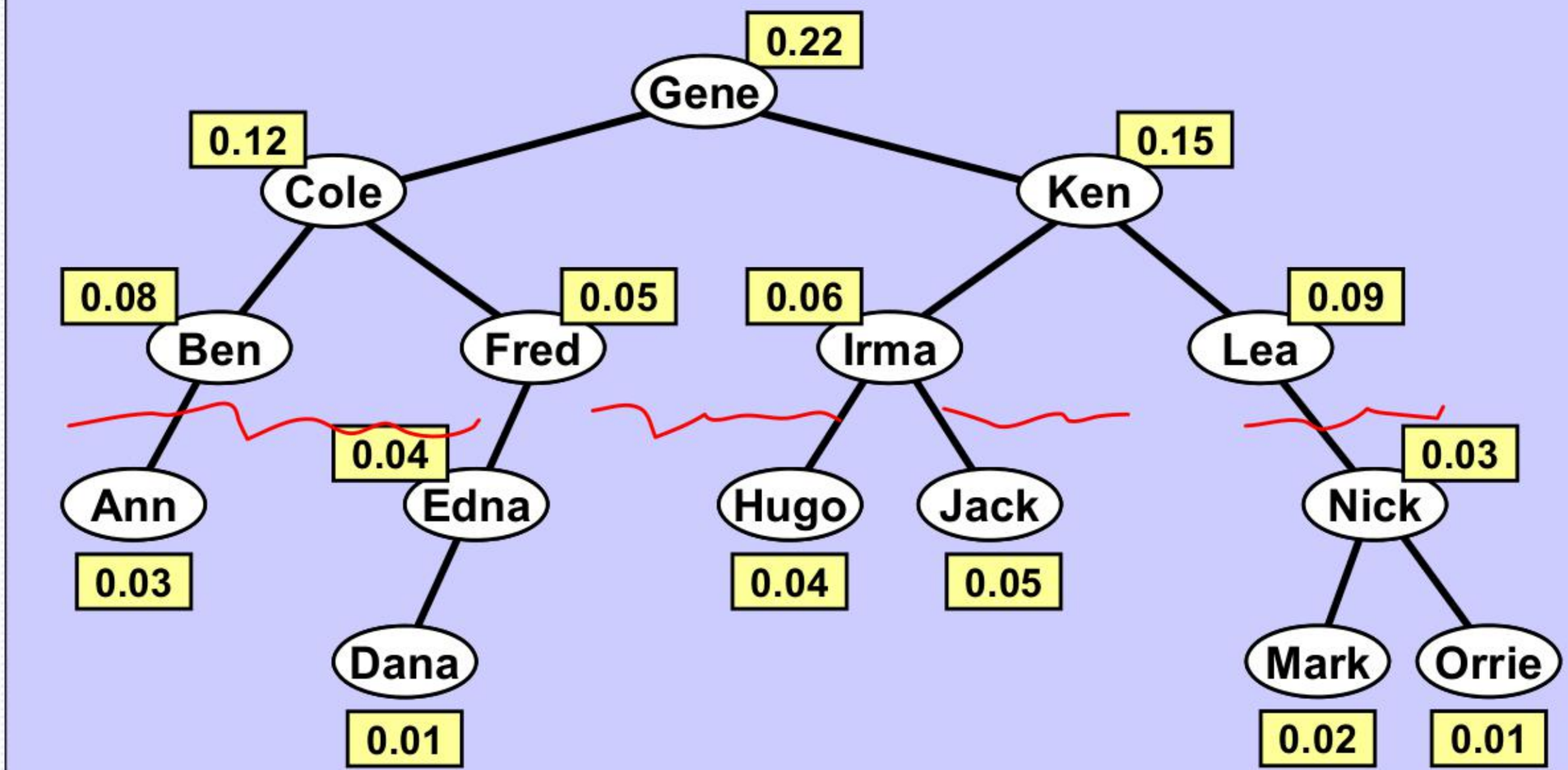
Optimální BVS

Struktura optimálního BVS s danými pravděpodobnostmi



Optimální BVS

Struktura optimálního BVS s danými pravděpodobnostmi



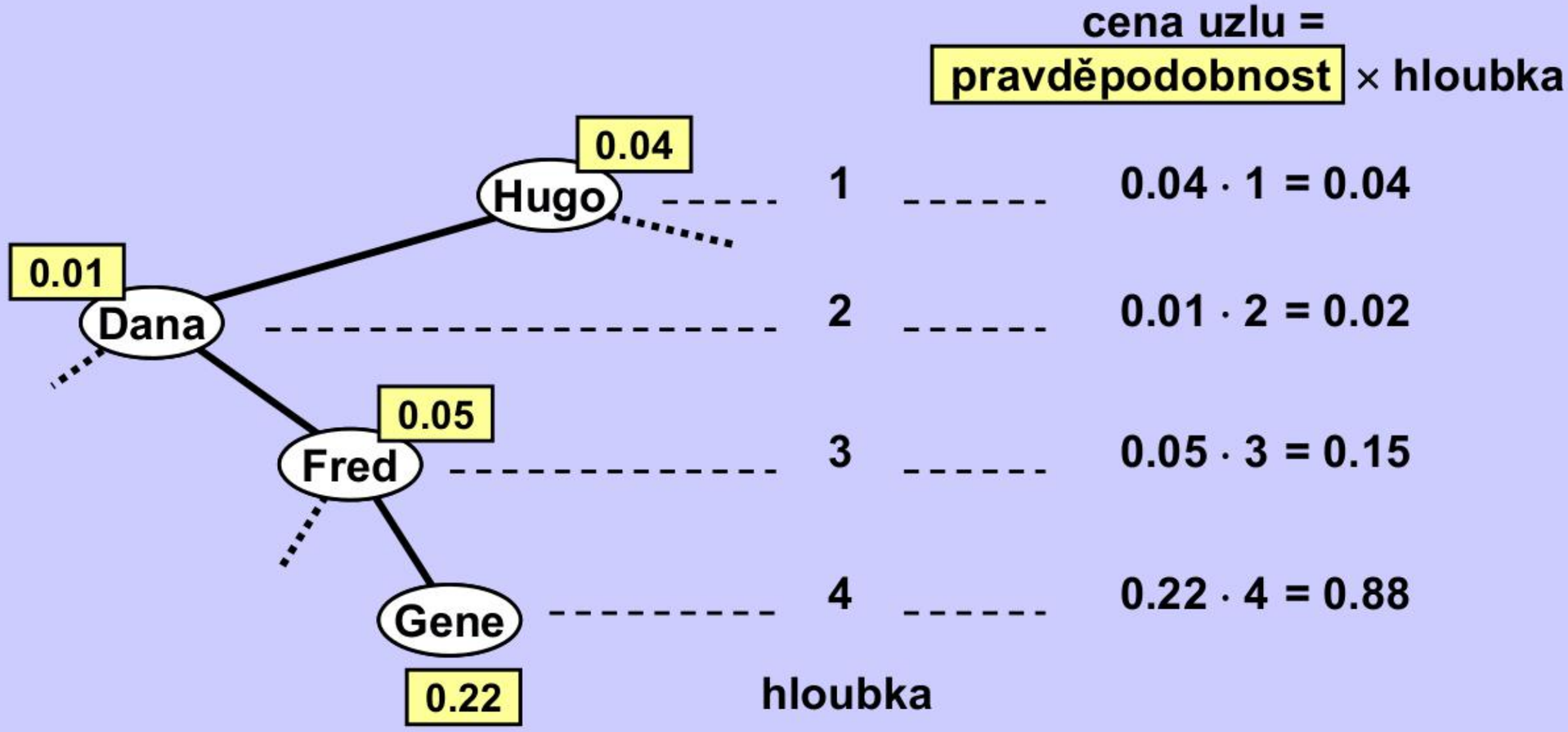
Cena vyváženého stromu

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	4	0.12 · 4 = 0.48
Dana	0.01	2	0.01 · 2 = 0.02
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	4	0.22 · 4 = 0.88
Hugo	0.04	1	0.04 · 1 = 0.04
Irma	0.06	4	0.06 · 4 = 0.24
Jack	0.05	3	0.05 · 3 = 0.15
Ken	0.15	4	0.15 · 4 = 0.60
Lea	0.09	2	0.09 · 2 = 0.18
Mark	0.02	4	0.02 · 4 = 0.08
Nick	0.03	3	0.03 · 3 = 0.09
Orrie	0.01	4	0.01 · 4 = 0.04
			Cena celkem: 3.47

Cena celkem = prům. poč. testů na jednu operaci Find.

Optimální binární vyhledávací strom

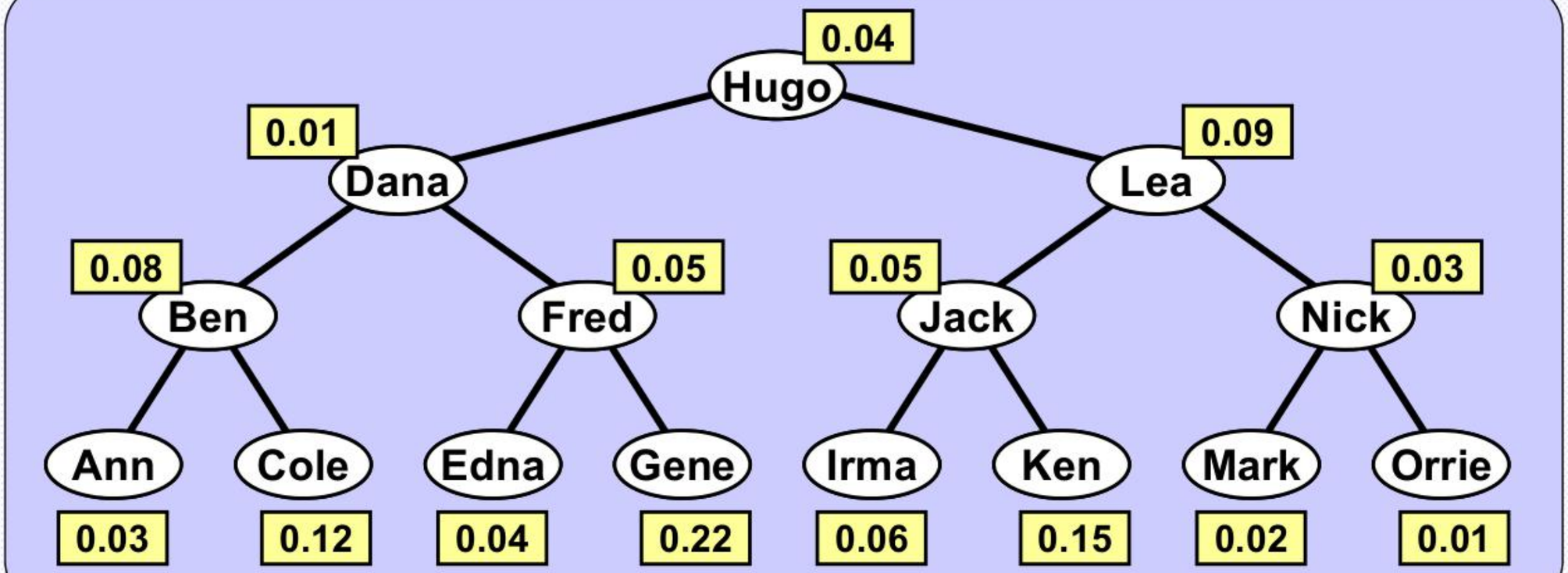
Cena jednotlivých uzlů v BVS



cena uzlu = průměrný počet testů na nalezení uzlu při jednom dotazu (Find)

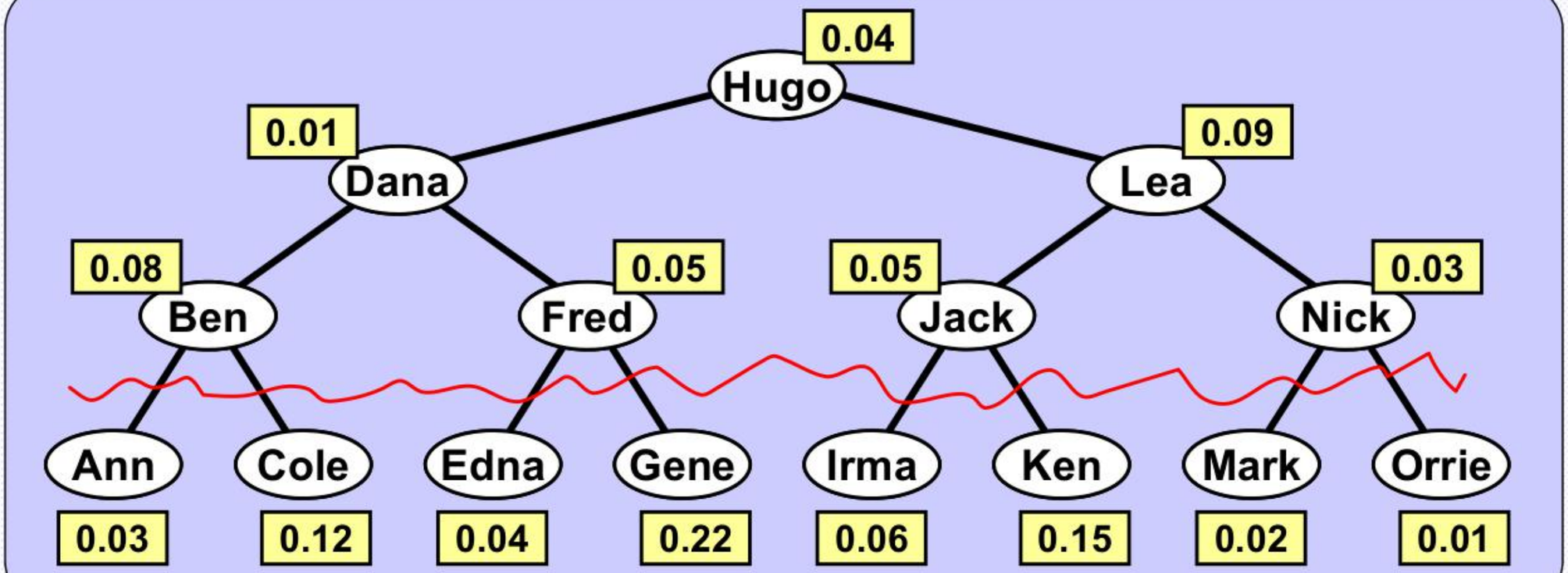
Optimální binární vyhledávací strom

Vyvážený, ale ne optimální



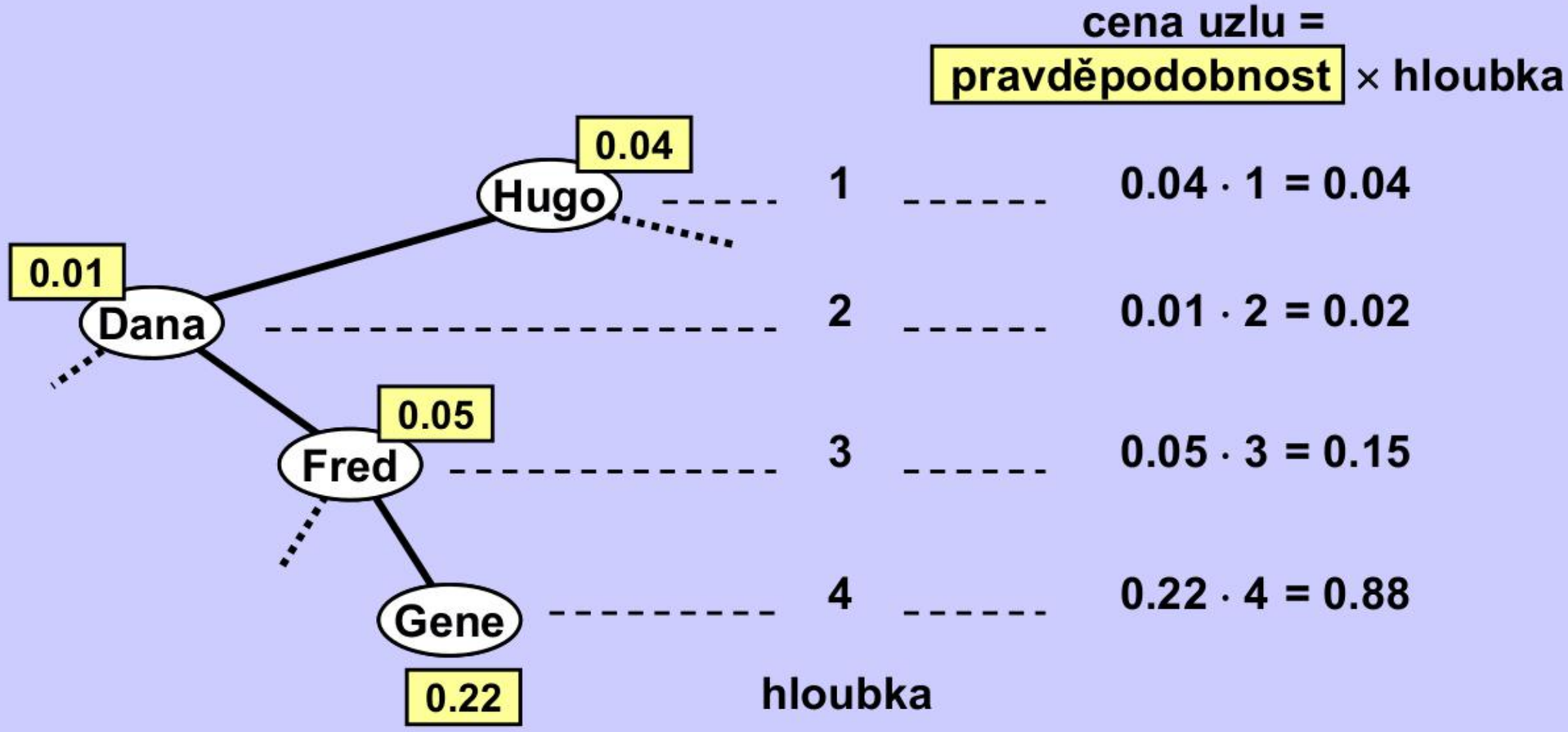
Optimální binární vyhledávací strom

Vyvážený, ale ne optimální



Optimální binární vyhledávací strom

Cena jednotlivých uzlů v BVS



cena uzlu = průměrný počet testů na nalezení uzlu při jednom dotazu (Find)

Cena vyváženého stromu

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	4	0.12 · 4 = 0.48
Dana	0.01	2	0.01 · 2 = 0.02
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	4	0.22 · 4 = 0.88
Hugo	0.04	1	0.04 · 1 = 0.04
Irma	0.06	4	0.06 · 4 = 0.24
Jack	0.05	3	0.05 · 3 = 0.15
Ken	0.15	4	0.15 · 4 = 0.60
Lea	0.09	2	0.09 · 2 = 0.18
Mark	0.02	4	0.02 · 4 = 0.08
Nick	0.03	3	0.03 · 3 = 0.09
Orrie	0.01	4	0.01 · 4 = 0.04
			Cena celkem: 3.47

Cena celkem = prům. poč. testů na jednu operaci Find.

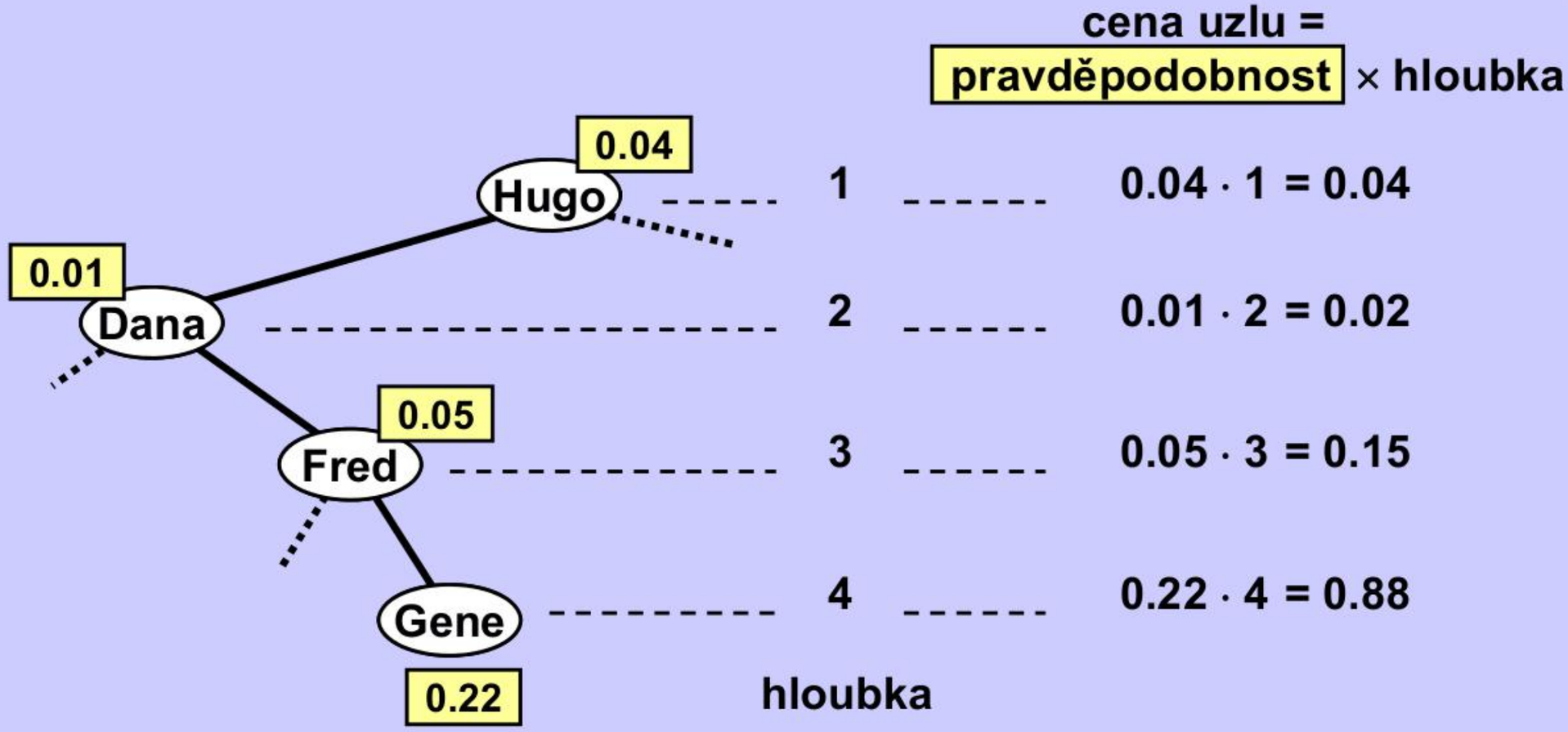
Cena vyváženého stromu

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	4	0.12 · 4 = 0.48
Dana	0.01	2	0.01 · 2 = 0.02
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	4	0.22 · 4 = 0.88
Hugo	0.04	1	0.04 · 1 = 0.04
Irma	0.06	4	0.06 · 4 = 0.24
Jack	0.05	3	0.05 · 3 = 0.15
Ken	0.15	4	0.15 · 4 = 0.60
Lea	0.09	2	0.09 · 2 = 0.18
Mark	0.02	4	0.02 · 4 = 0.08
Nick	0.03	3	0.03 · 3 = 0.09
Orrie	0.01	4	0.01 · 4 = 0.04
		Cena celkem:	3.47

Cena celkem = prům. poč. testů na jednu operaci Find.

Optimální binární vyhledávací strom

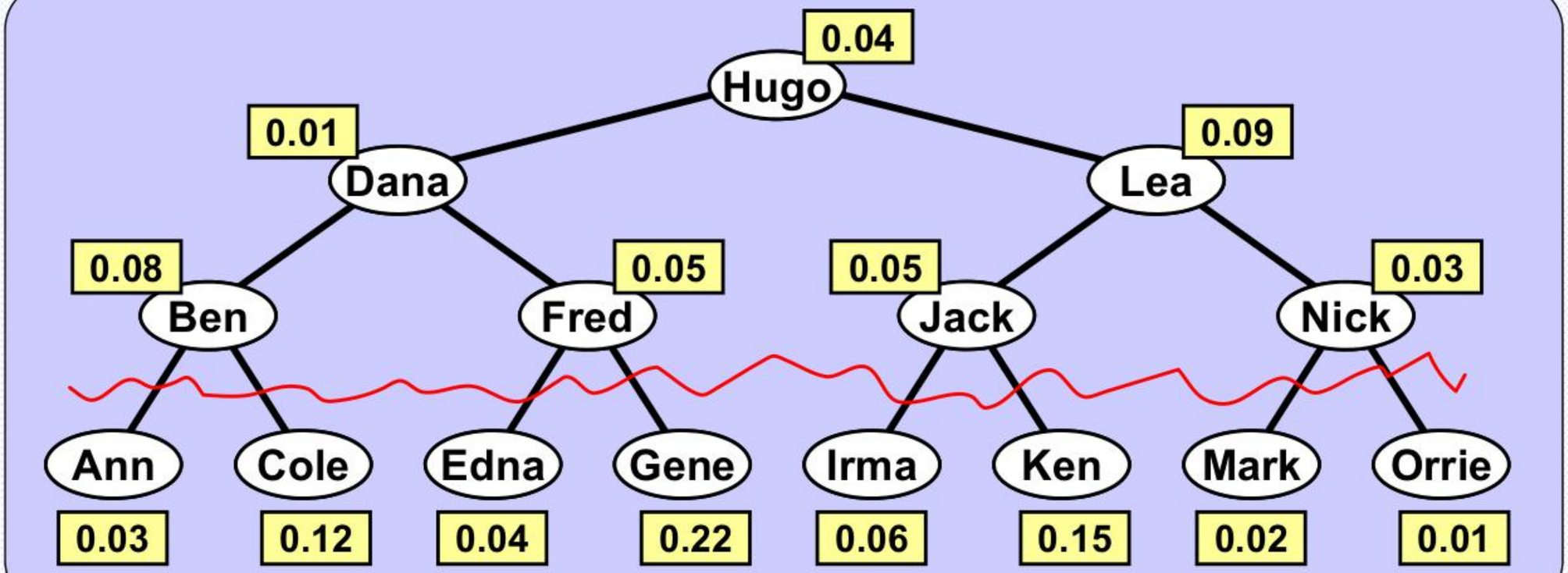
Cena jednotlivých uzlů v BVS



cena uzlu = průměrný počet testů na nalezení uzlu při jednom dotazu (Find)

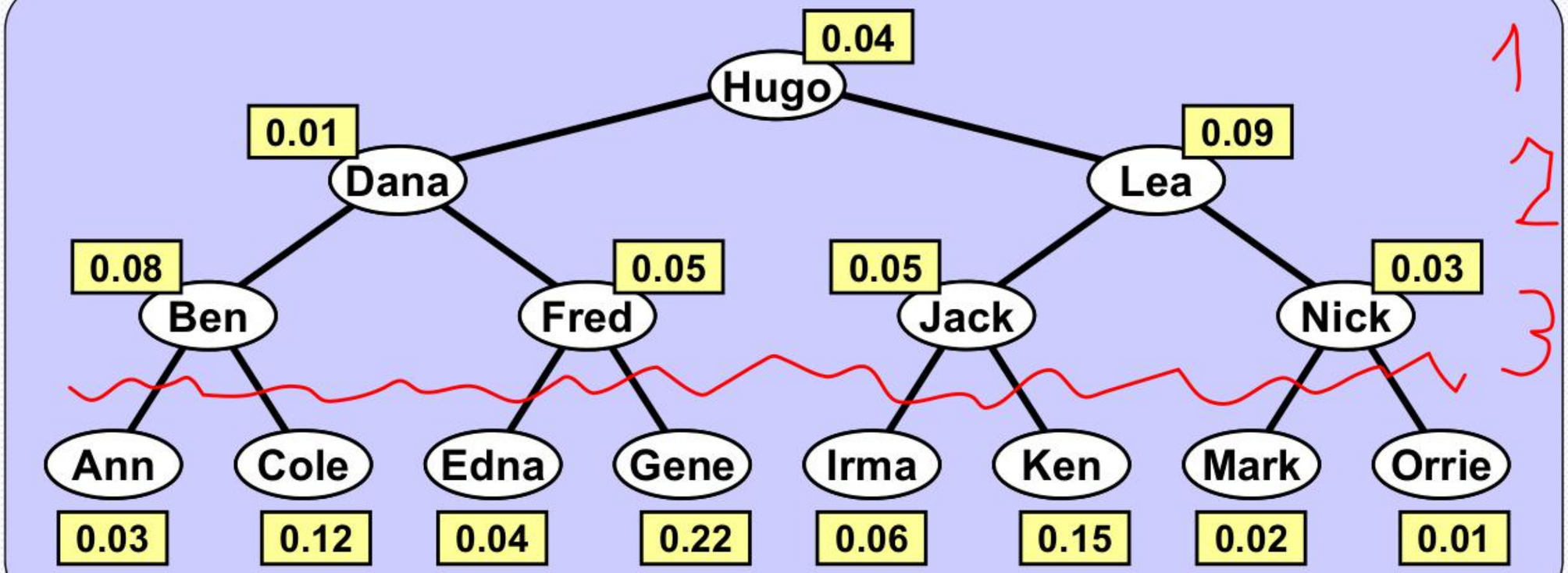
Optimální binární vyhledávací strom

Vyvážený, ale ne optimální



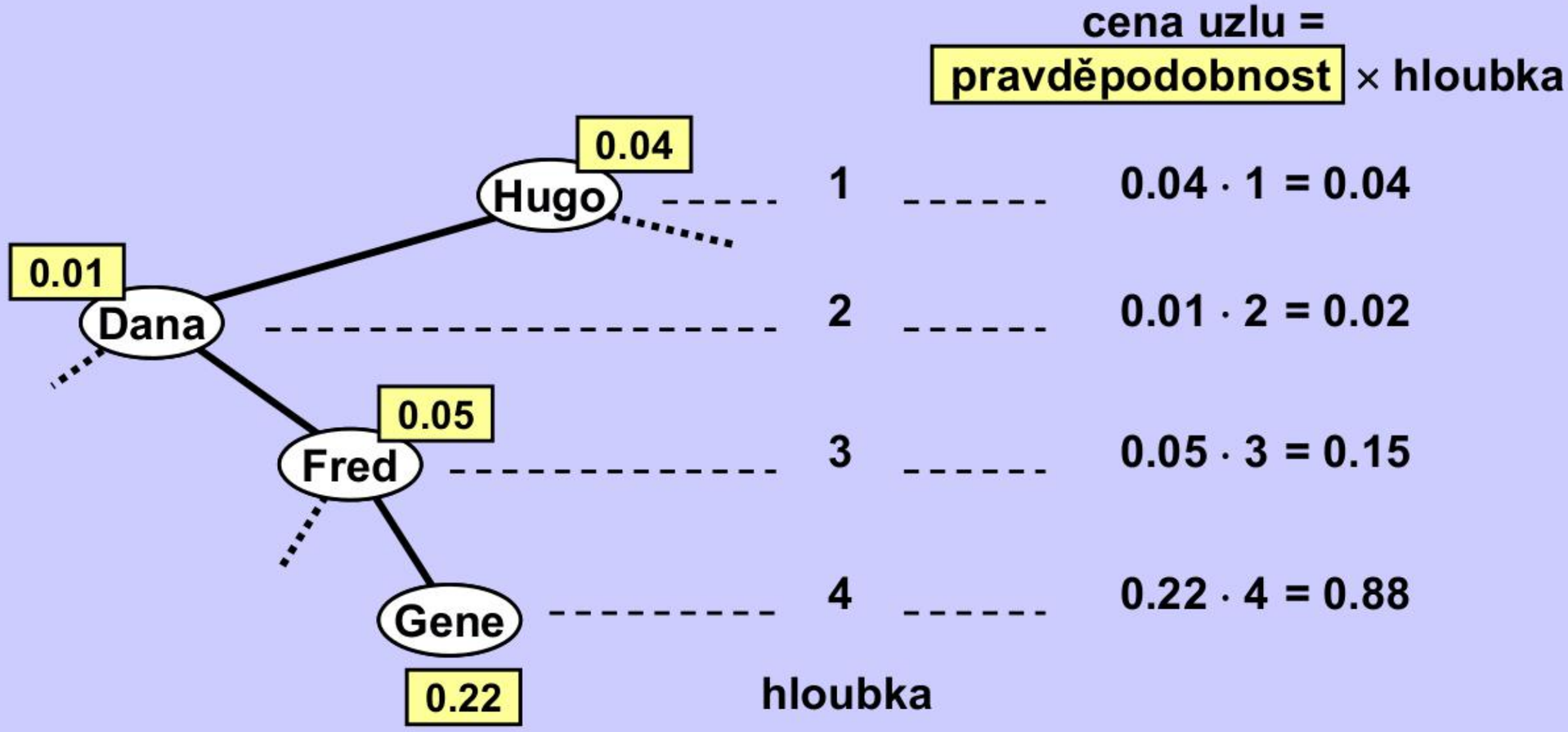
Optimální binární vyhledávací strom

Vyvážený, ale ne optimální



Optimální binární vyhledávací strom

Cena jednotlivých uzlů v BVS



cena uzlu = průměrný počet testů na nalezení uzlu při jednom dotazu (Find)

Cena vyváženého stromu

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	4	0.12 · 4 = 0.48
Dana	0.01	2	0.01 · 2 = 0.02
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	4	0.22 · 4 = 0.88
Hugo	0.04	1	0.04 · 1 = 0.04
Irma	0.06	4	0.06 · 4 = 0.24
Jack	0.05	3	0.05 · 3 = 0.15
Ken	0.15	4	0.15 · 4 = 0.60
Lea	0.09	2	0.09 · 2 = 0.18
Mark	0.02	4	0.02 · 4 = 0.08
Nick	0.03	3	0.03 · 3 = 0.09
Orrie	0.01	4	0.01 · 4 = 0.04
		Cena celkem:	3.47

Cena celkem = prům. poč. testů na jednu operaci Find.

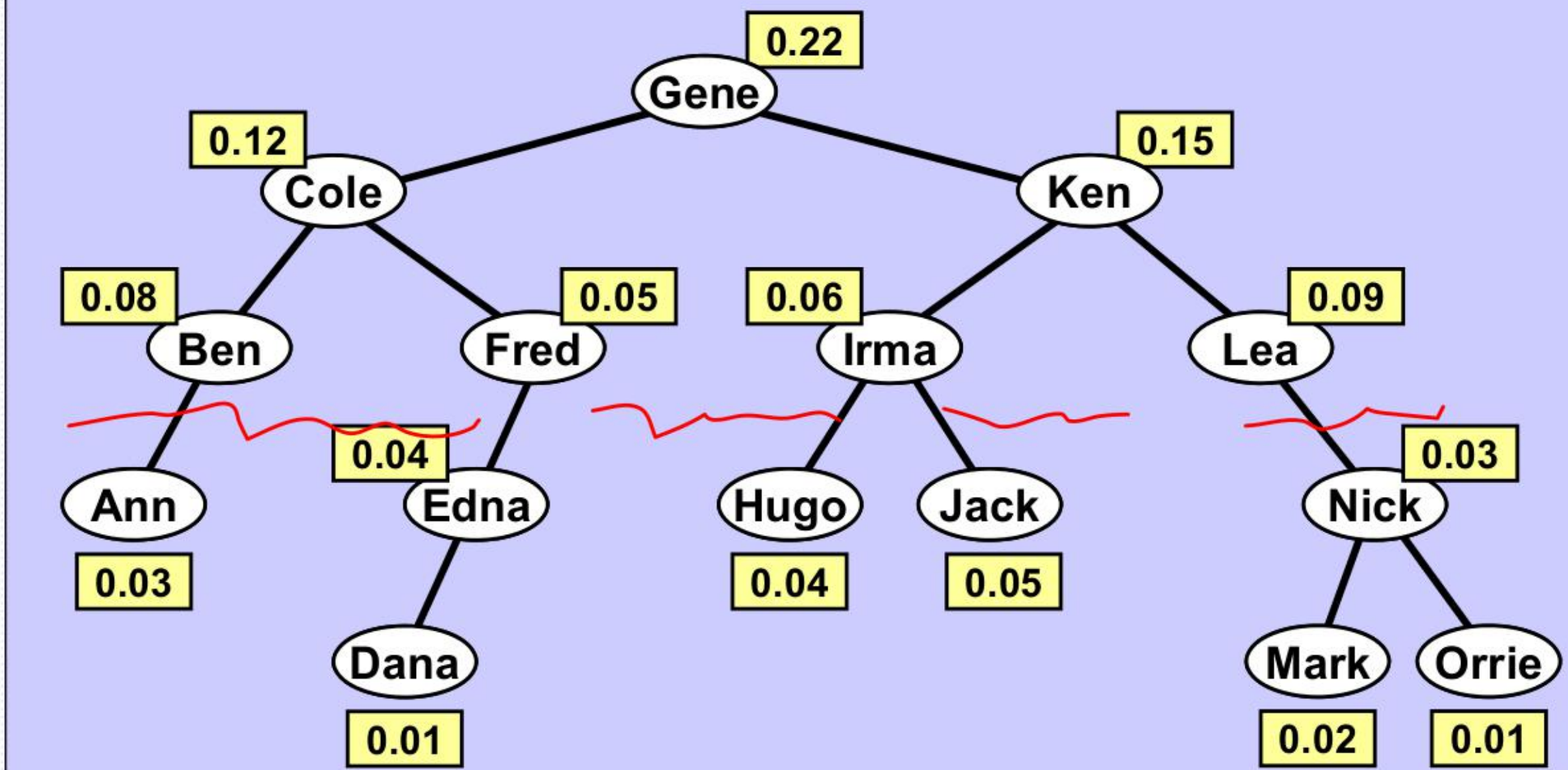
Cena vyváženého stromu

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	4	0.12 · 4 = 0.48
Dana	0.01	2	0.01 · 2 = 0.02
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	4	0.22 · 4 = 0.88
Hugo	0.04	1	0.04 · 1 = 0.04
Irma	0.06	4	0.06 · 4 = 0.24
Jack	0.05	3	0.05 · 3 = 0.15
Ken	0.15	4	0.15 · 4 = 0.60
Lea	0.09	2	0.09 · 2 = 0.18
Mark	0.02	4	0.02 · 4 = 0.08
Nick	0.03	3	0.03 · 3 = 0.09
Orrie	0.01	4	0.01 · 4 = 0.04
		Cena celkem:	3.47

Cena celkem = prům. poč. testů na jednu operaci Find.

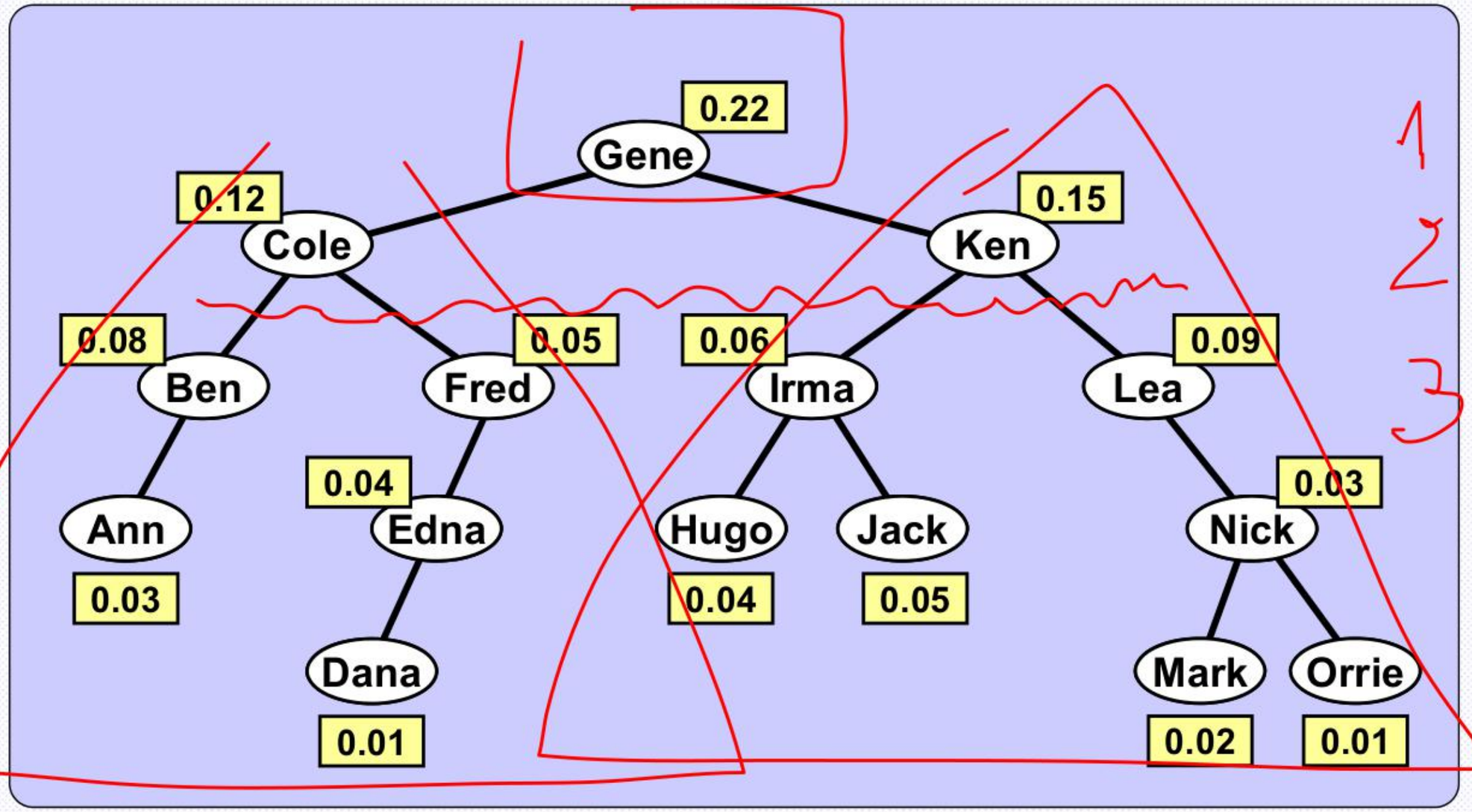
Optimální BVS

Struktura optimálního BVS s danými pravděpodobnostmi



Optimální BVS

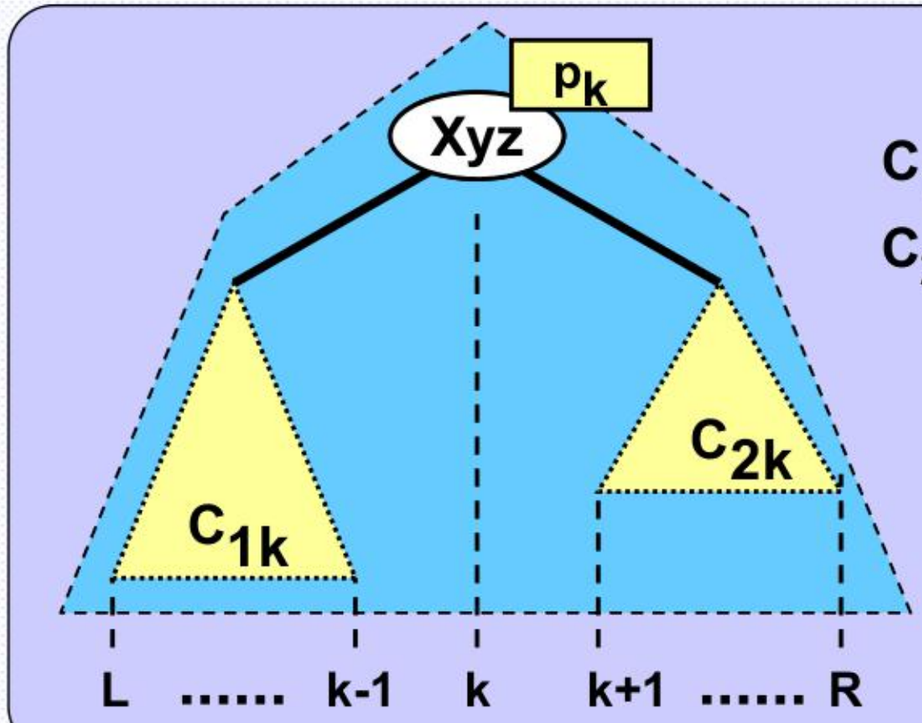
Struktura optimálního BVS s danými pravděpodobnostmi



Cena optimálního BVS

klíč	pravděp. p_k	hloubka d_k	$p_k \cdot d_k$
Ann	0.03	4	0.03 · 4 = 0.12
Ben	0.08	3	0.08 · 3 = 0.24
Cole	0.12	2	0.12 · 2 = 0.24
Dana	0.01	5	0.01 · 5 = 0.05
Edna	0.04	4	0.04 · 4 = 0.16
Fred	0.05	3	0.05 · 3 = 0.15
Gene	0.22	1	0.22 · 1 = 0.22
Hugo	0.04	4	0.04 · 4 = 0.16
Irma	0.06	3	0.06 · 3 = 0.18
Jack	0.05	4	0.05 · 4 = 0.20
Ken	0.15	2	0.15 · 2 = 0.30
Lea	0.09	3	0.09 · 3 = 0.27
Mark	0.02	5	0.02 · 5 = 0.10
Nick	0.03	4	0.03 · 4 = 0.12
Orrie	0.01	5	0.01 · 5 = 0.05
			Cena celkem 2.56
Zrychlení 3.47 : 2.56 = 1 : 0.74			

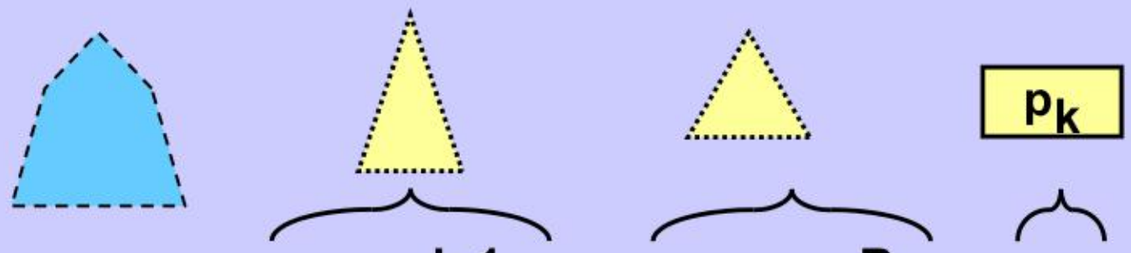
Výpočet ceny optimálního BVS



C_{1k} ….. cena levého podstromu uzlu k

C_{2k} ….. Cena pravého podstromu uzlu k

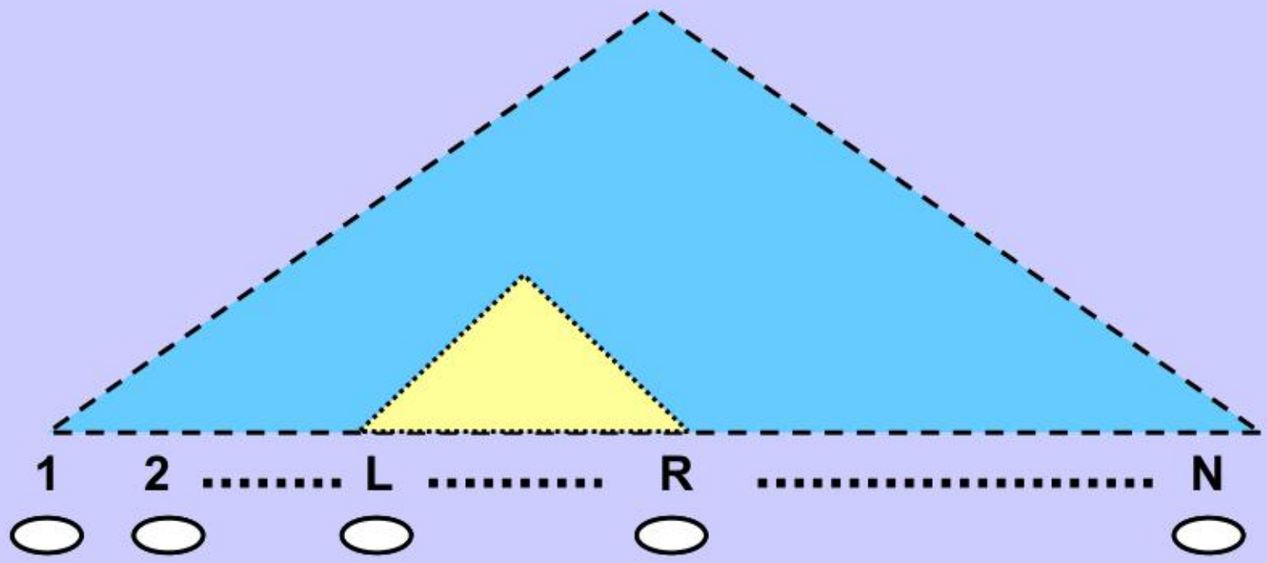
Rekurzivní myšlenka



$$\text{Cena} = C_{1k} + \sum_{i=L}^{k-1} p_i + C_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

Výpočet ceny optimálního BVS

Malé optimální podstromy



Nad prvky s indexy od L do R
lze jistě vytvořit jeden optimální podstrom.

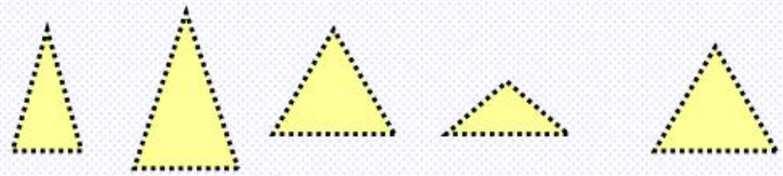
Velikost stromu
= poč. uzlů
= L-R+1

Máme	N	optimalních podstromů velikosti	1
	N-1		2
	N-2		3
	⋮		⋮
	1	podstrom	N

Celkem máme N * (N+1) /2 různých optimálních podstromů.

Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R

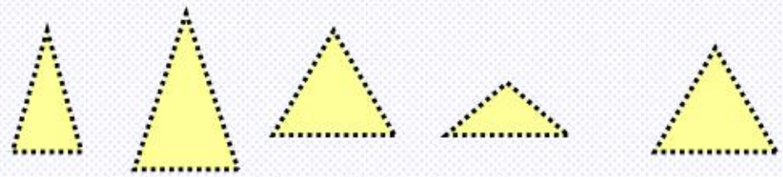
3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$C_{1k} + \sum_{i=L}^{k-1} p_i + C_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

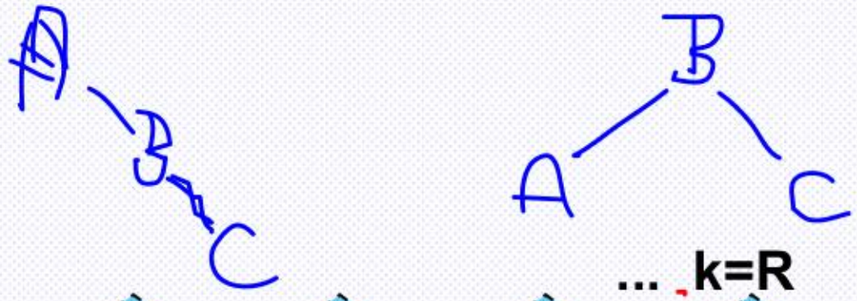
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

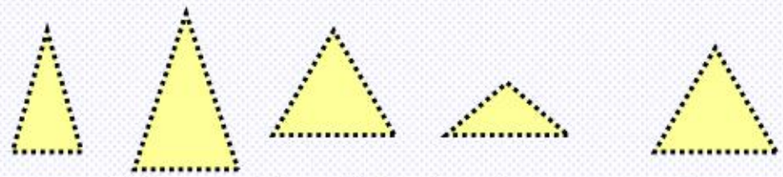
$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

$$(*) \quad = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

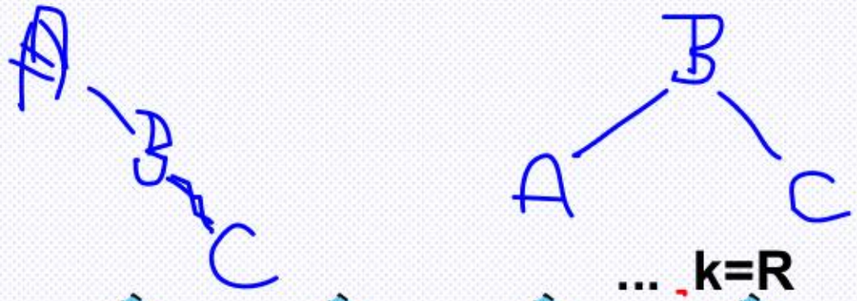
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



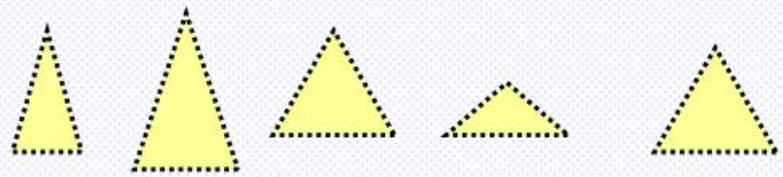
3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

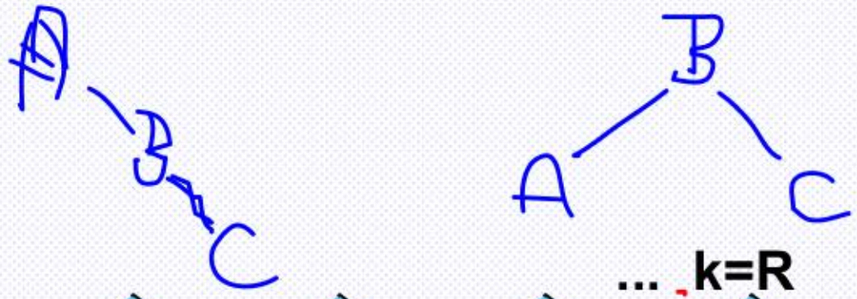
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



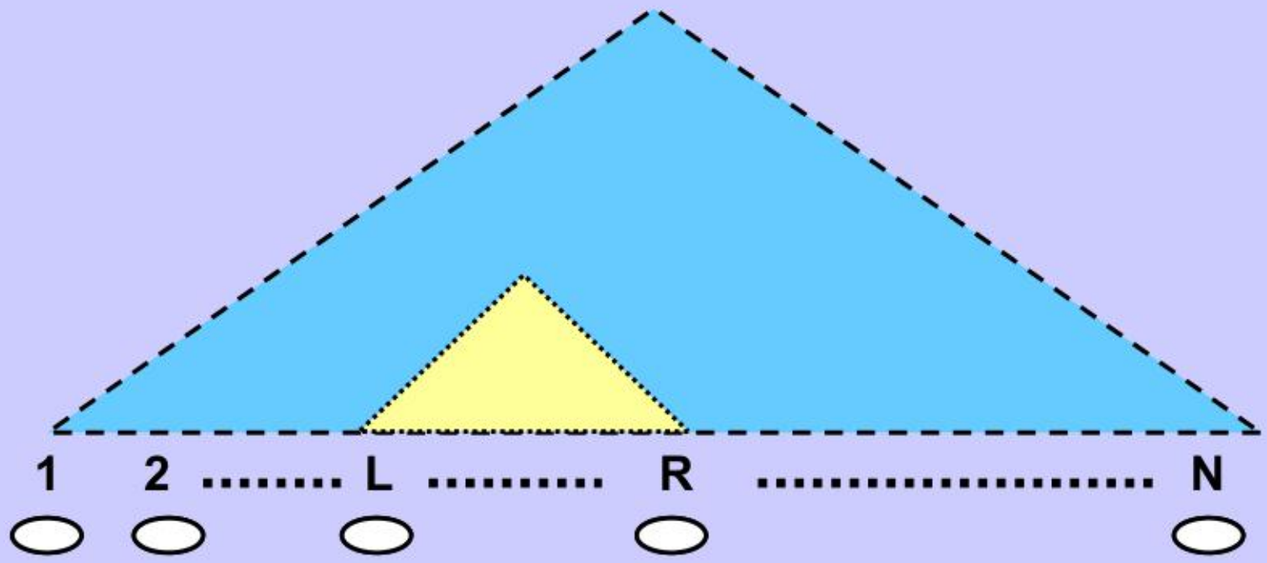
3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

Výpočet ceny optimálního BVS

Malé optimální podstromy



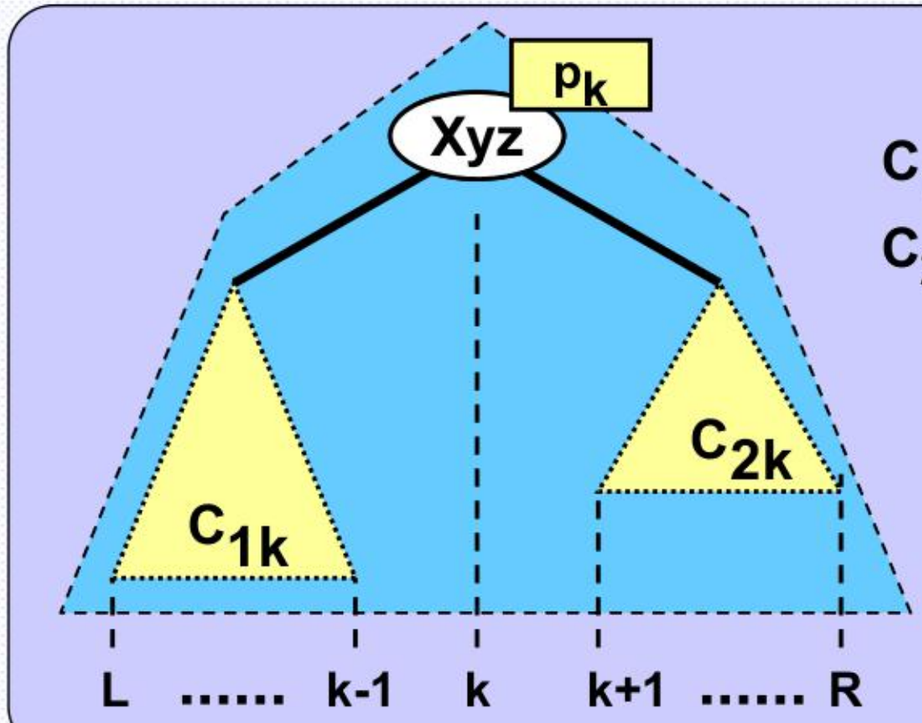
Nad prvky s indexy od L do R
lze jistě vytvořit jeden optimální podstrom.

Velikost stromu
= poč. uzlů
= L-R+1

Máme	N	optimalních podstromů velikosti	1
	N-1		2
	N-2		3
	⋮		⋮
	1	podstrom	N

Celkem máme N * (N+1) /2 různých optimálních podstromů.

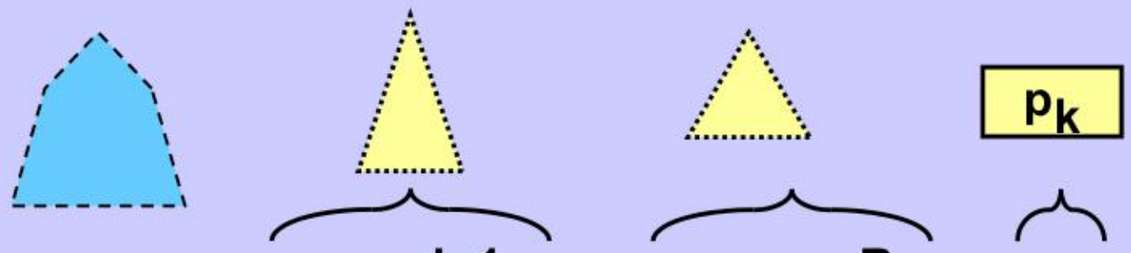
Výpočet ceny optimálního BVS



C_{1k} ….. cena levého podstromu uzlu k

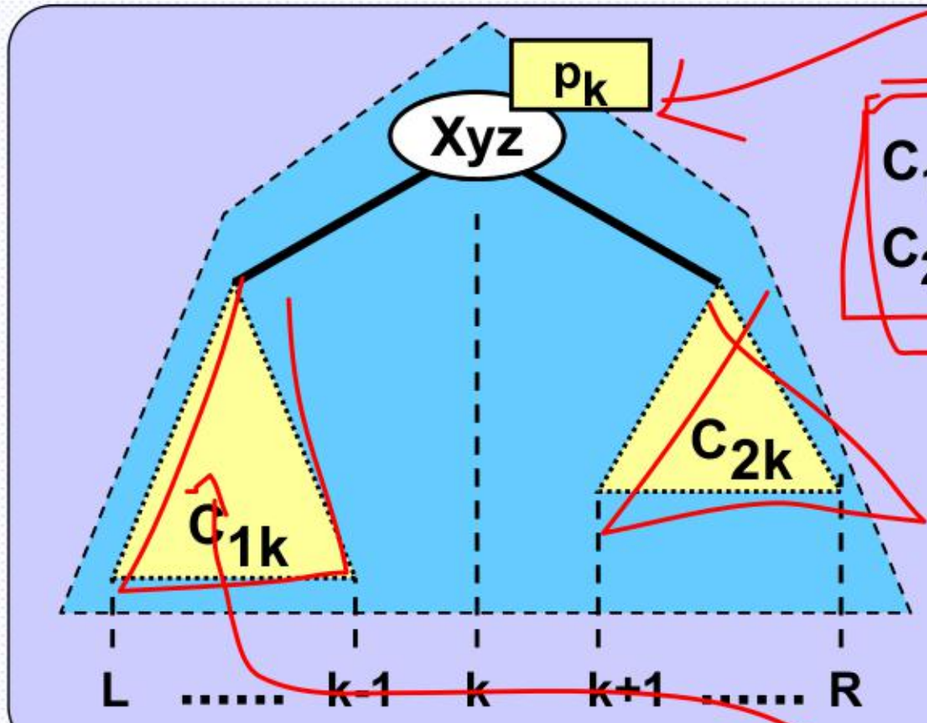
C_{2k} ….. Cena pravého podstromu uzlu k

Rekurzivní myšlenka



$$\text{Cena} = C_{1k} + \sum_{i=L}^{k-1} p_i + C_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

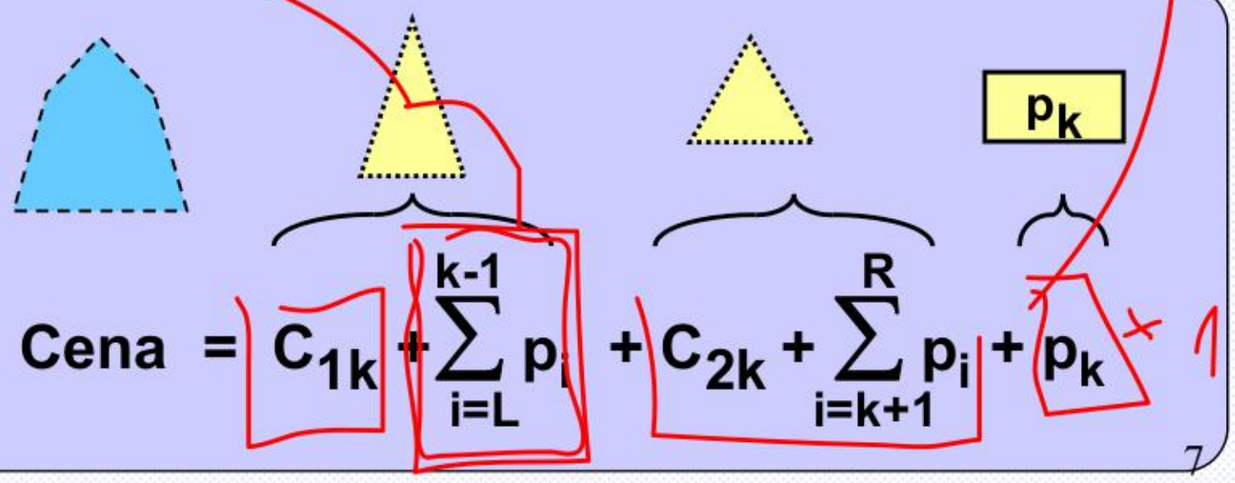
Výpočet ceny optimálního BVS



C_{1k} ..... cena levého podstromu uzlu k

C_{2k} ..... Cena pravého podstromu uzlu k

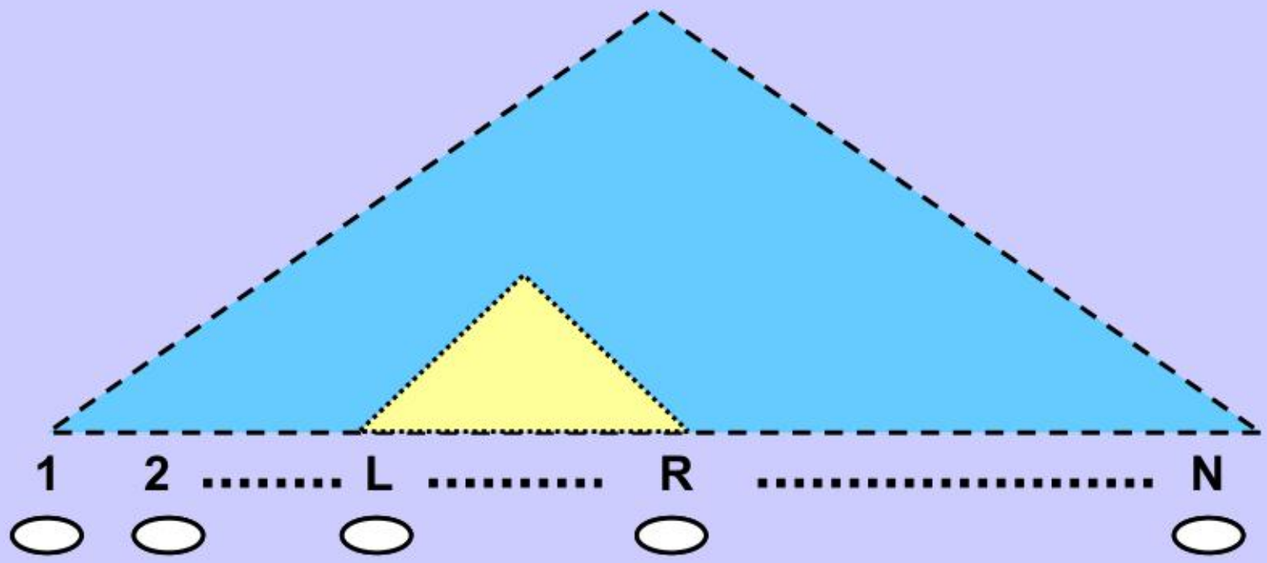
Rekurzivní myšlenka



$$\text{Cena} = C_{1k} + \sum_{i=L}^{k-1} p_i + C_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

Výpočet ceny optimálního BVS

Malé optimální podstromy



Nad prvky s indexy od L do R
lze jistě vytvořit jeden optimální podstrom.

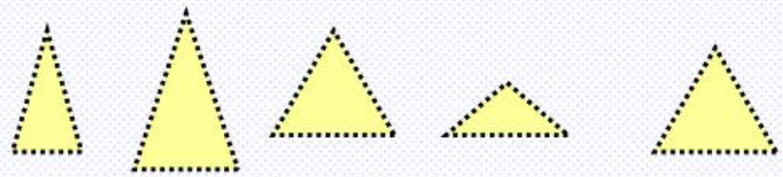
Velikost stromu
= poč. uzlů
= L-R+1

Máme	N	optimalních podstromů velikosti	1
	N-1		2
	N-2		3
	⋮		⋮
	1	podstrom	N

Celkem máme N * (N+1) /2 různých optimálních podstromů.

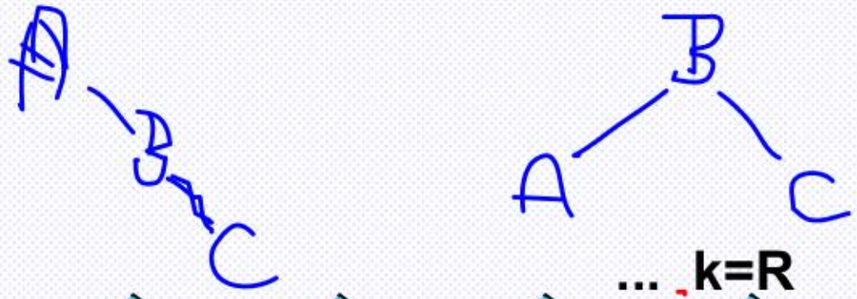
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



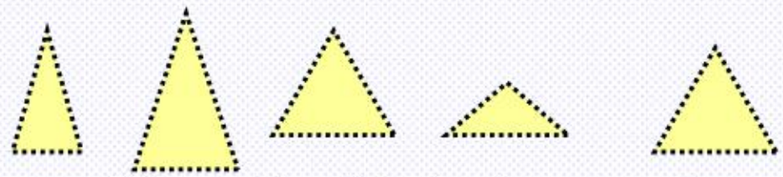
3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

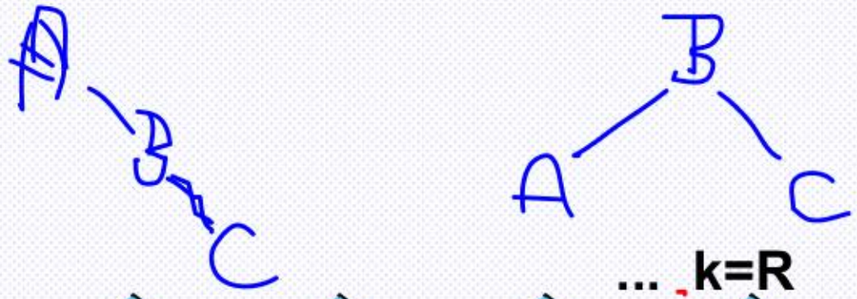
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

$$(*) \quad = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

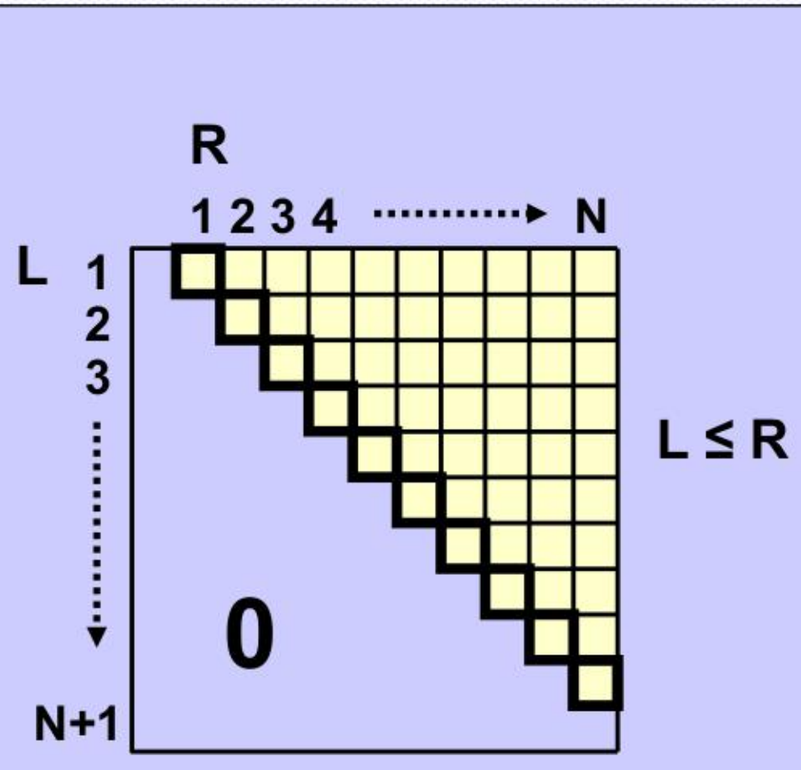
$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Datové struktury pro výpočet optmálního BVS

Ceny optimálních podstromů

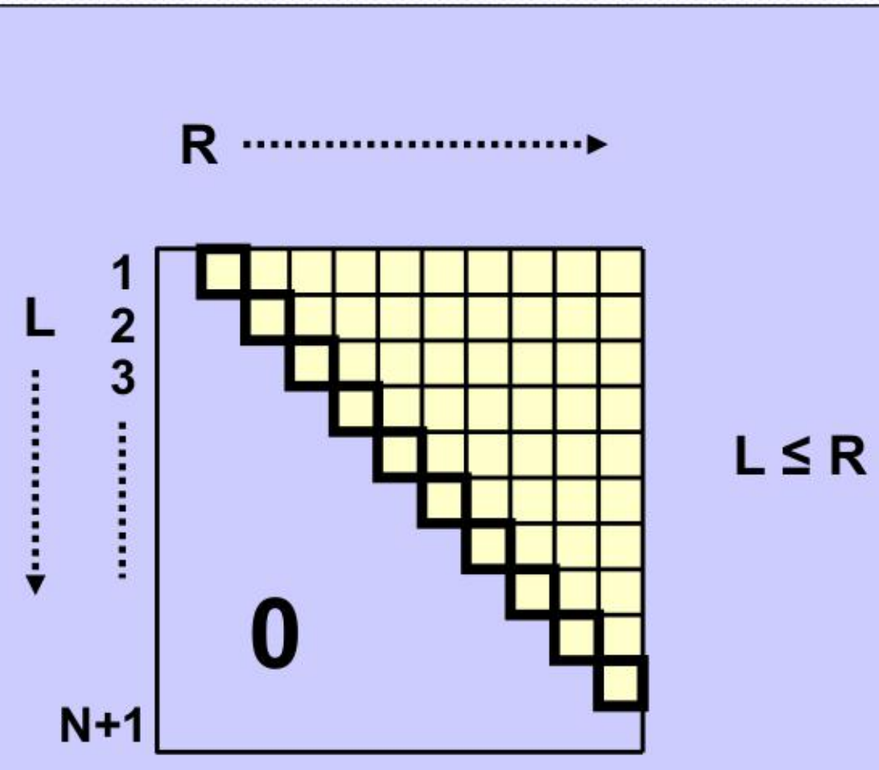
pole C [L][R] (L ≤ R)



diagonála ... L= R

Kořeny optimálních podstromů

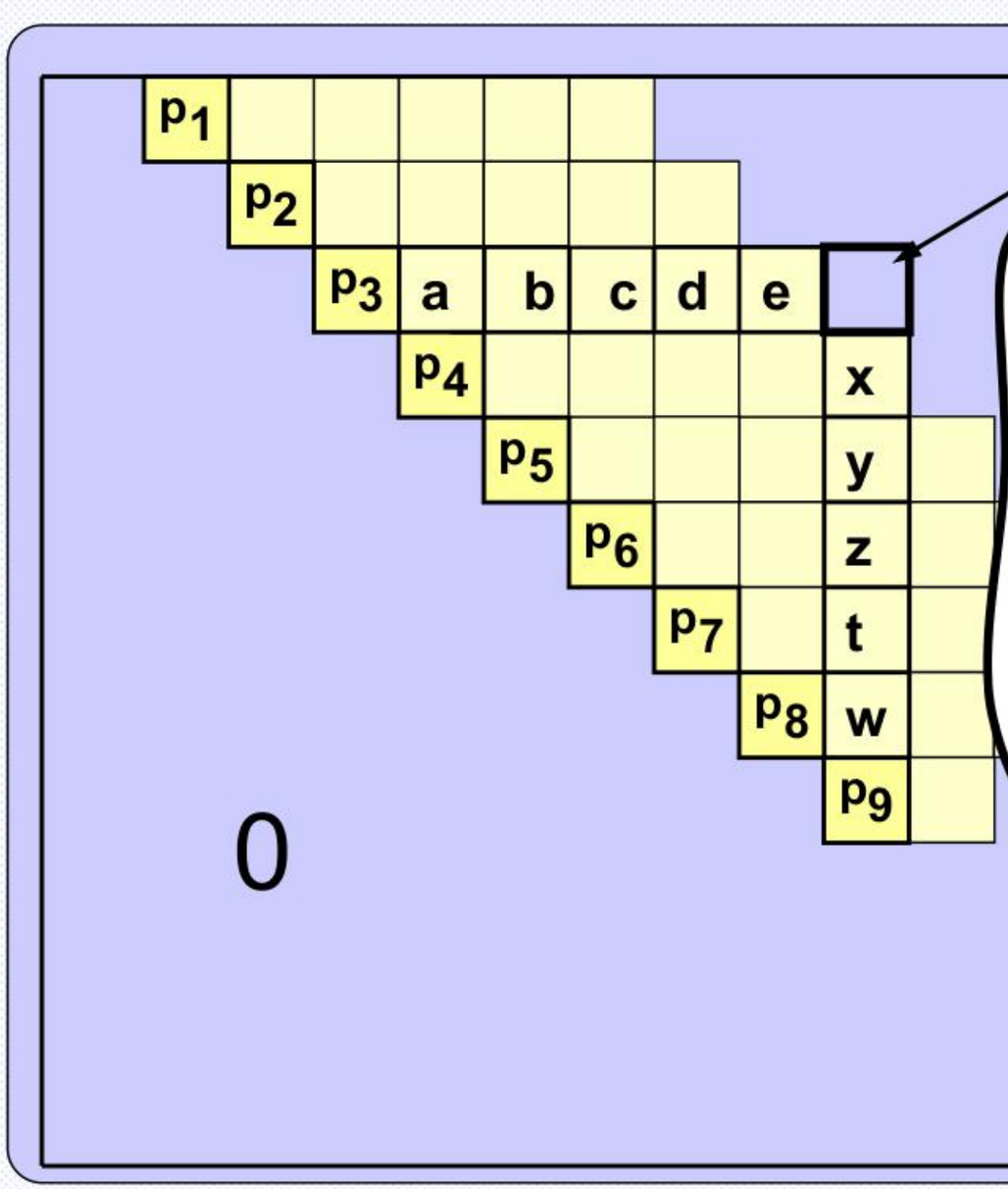
pole roots [L][R] (L ≤ R)



diagonála ... L= R

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



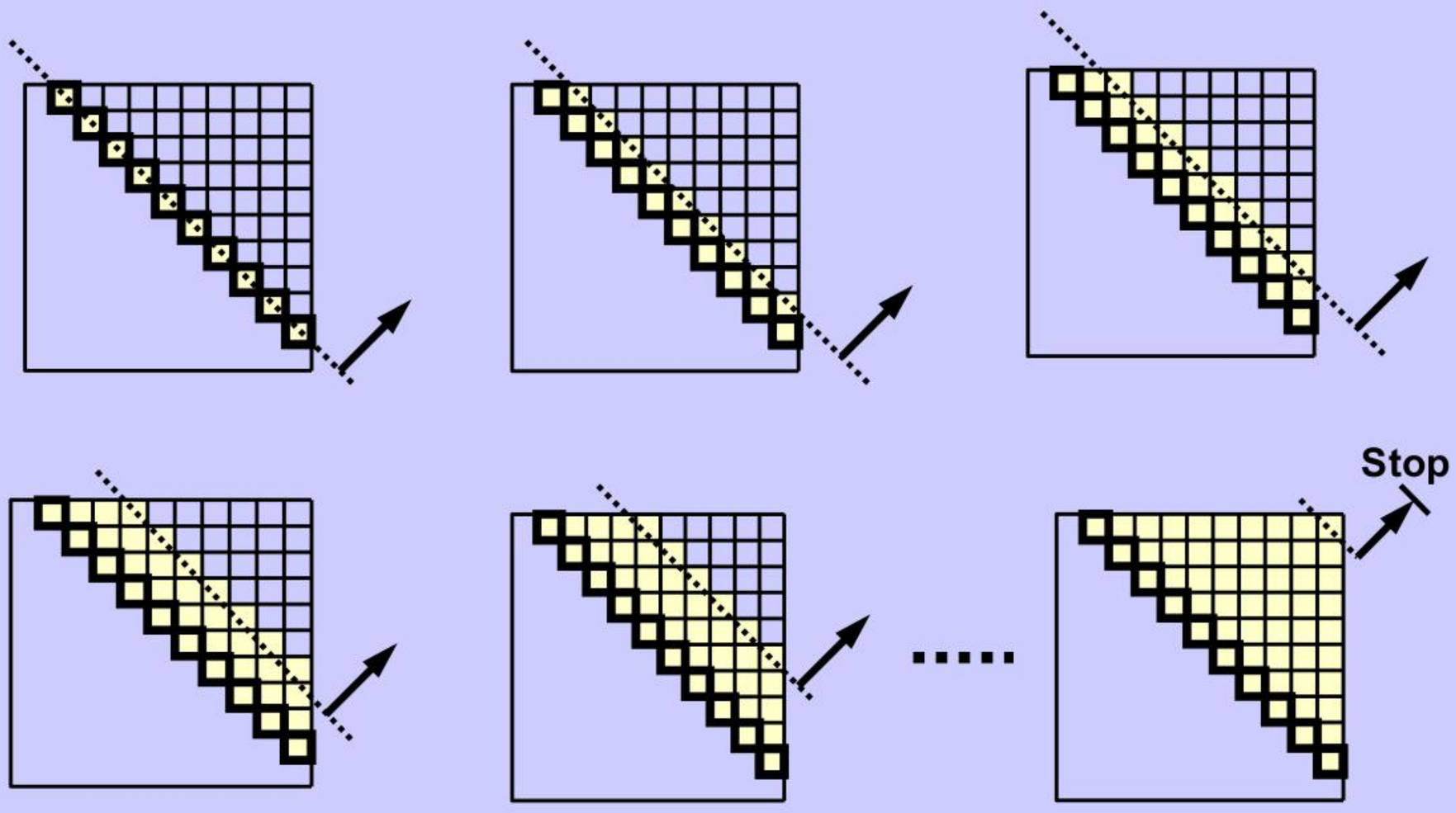
L=3, R=9

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Výpočet optimálního BVS

Strategie DP
– nejprve se zpracují nejmenší podstromy, pak větší, atd…



Výpočet optimálního BVS

Výpočet DP tabulek cen a kořenů

```
void optimalTree() {
  int L, R; double min;

  // size = 1
  for( i=0; i<=N; i++ ) {
    C[i][i] = pravděpodobnost[i]; roots[i][i] = i;

  // size > 1
  for( int size = 2; size <= N; size++ ) {
    L = 1; R = size;
    while( R <= N ) {
      C[L][R] = min(C[L][k-1]+C[k+1][R], k = L..R);
      roots[L][R] = 'k minimalizující předch. řádek';
      C[L][R] += sum(C[i][i], i = L..R);
      L++; R++;
} } }
```

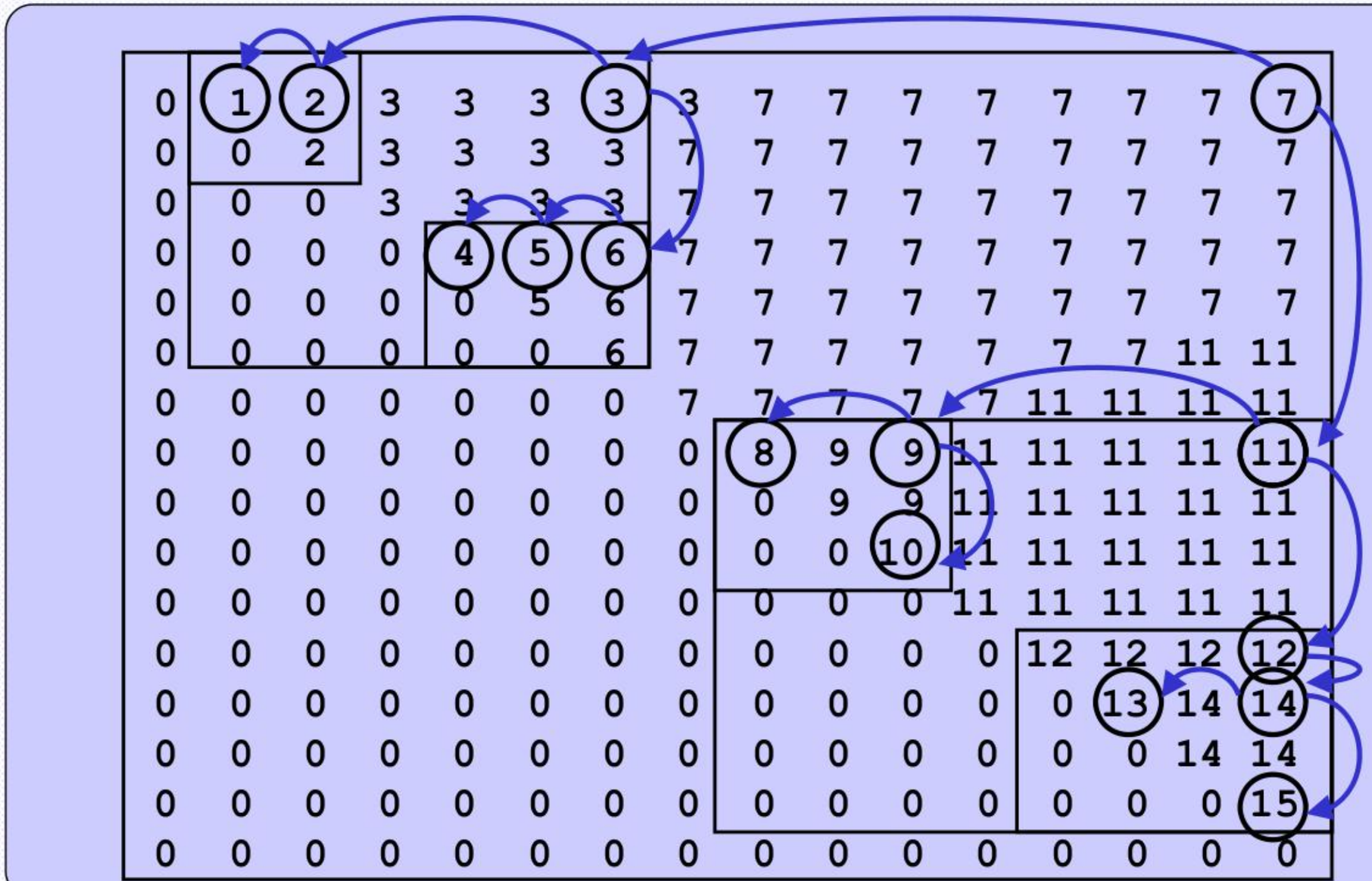
Výpočet optimálního BVS

Vybudování optimálního stromu pomocí rekonstrukční tabulky kořenů

```
void buildTree( int L, int R) {

  if (R < L) return;

  int keyIndex = roots[L][R];
  // keys ... sorted array of keys
  int key = keys[roots[L][R]];

  insert(root, key);     // standard BST insert
  buildTree( L, keyIndex -1 );
  buildTree( keyIndex +1, R );
}
```

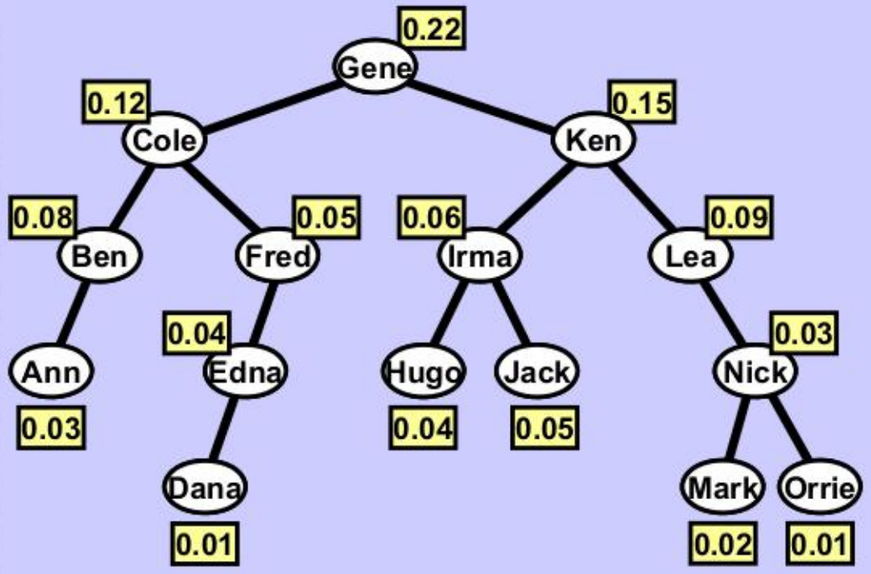
Výpočet optimálního BVS

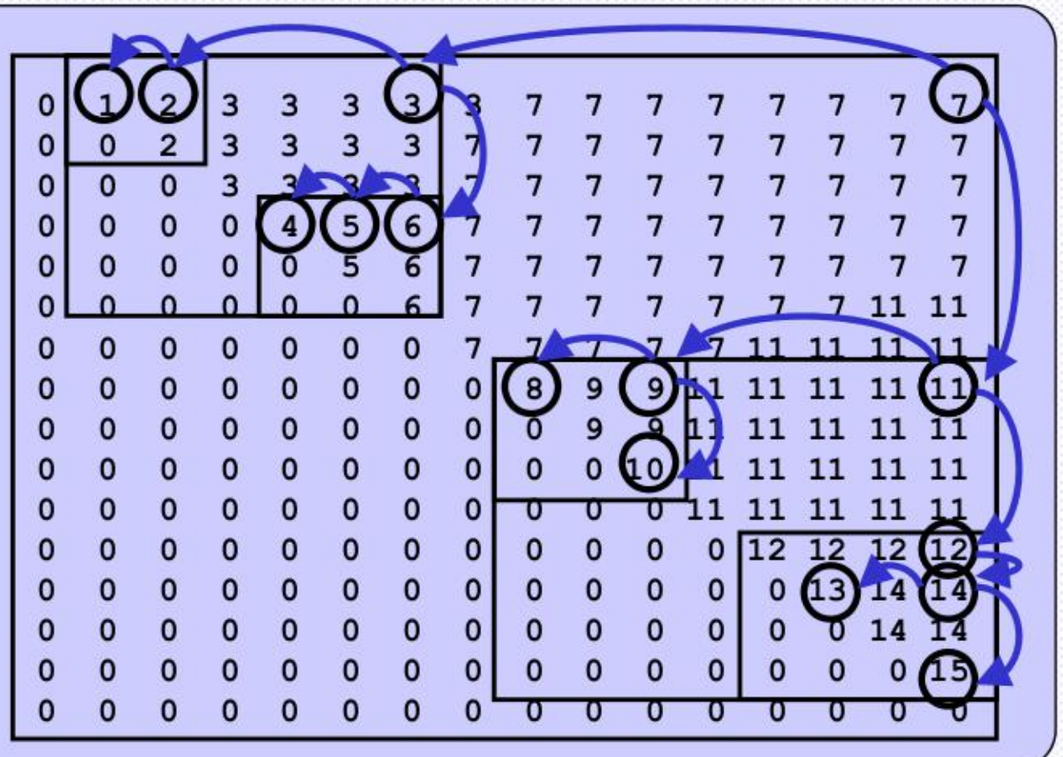
Kořeny optimálních podstromů

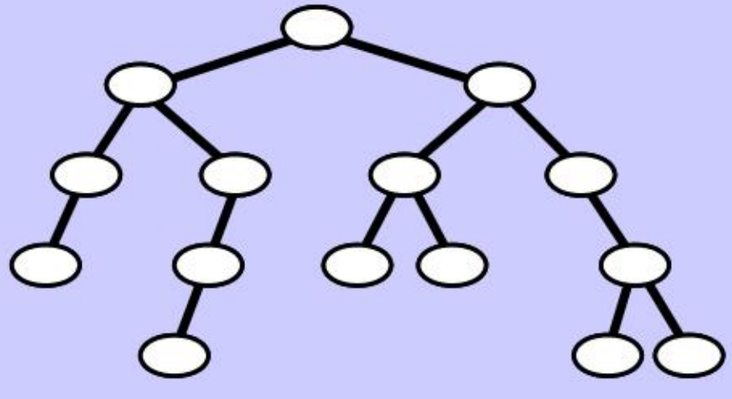


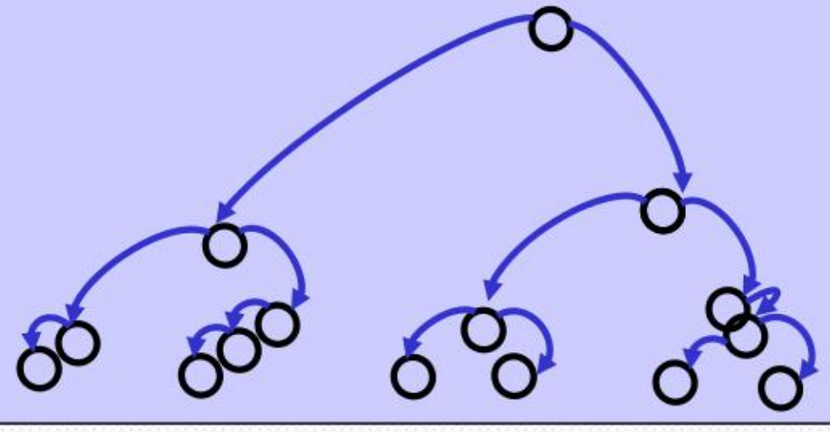
Výpočet optimálního BVS

Korespondence stromů









Výpočet optimálního BVS

Ceny optimálních podstromů

	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

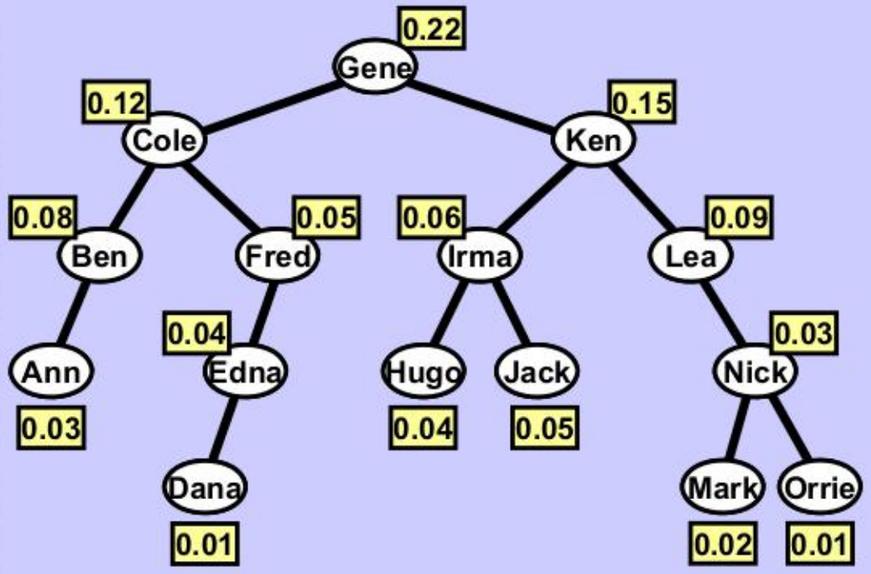
Výpočet optimálního BVS

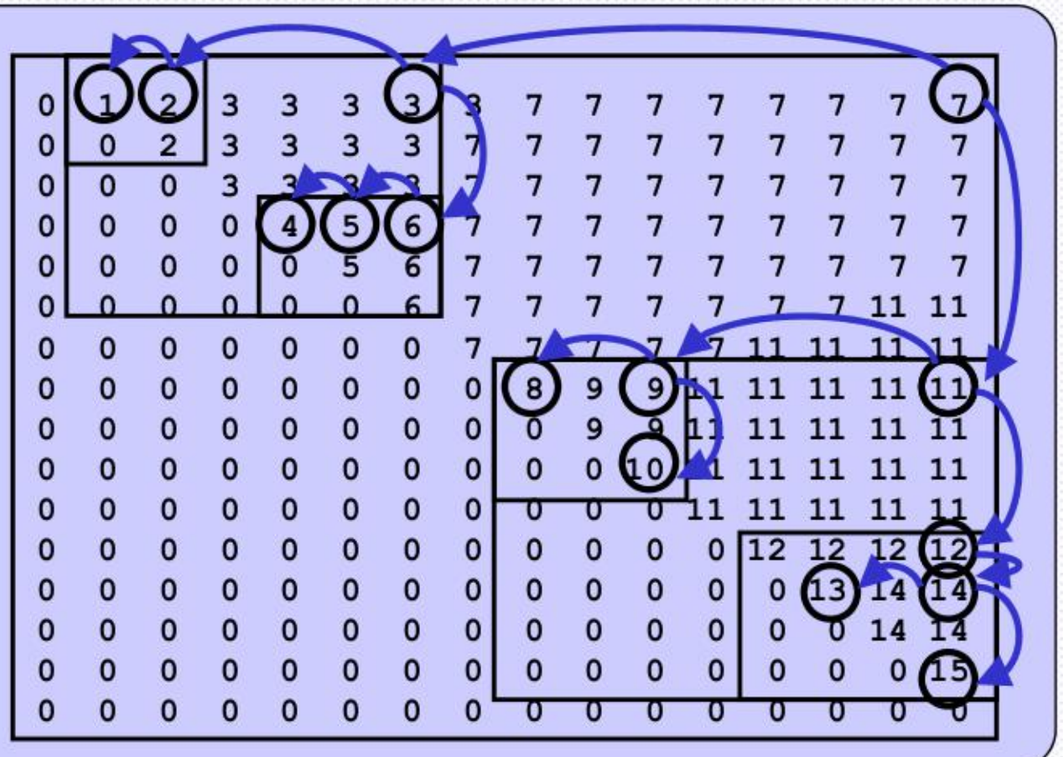
Ceny optimálních podstromů

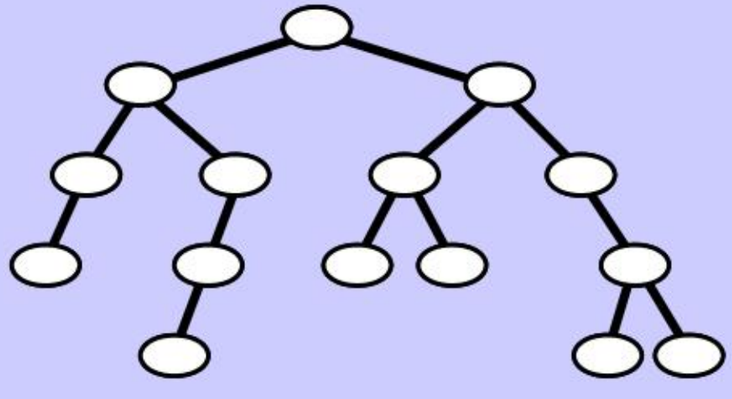
	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

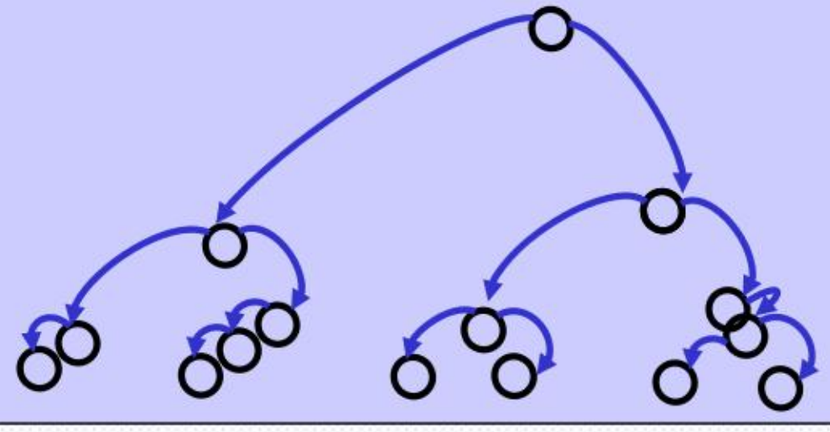
Výpočet optimálního BVS

Korespondence stromů



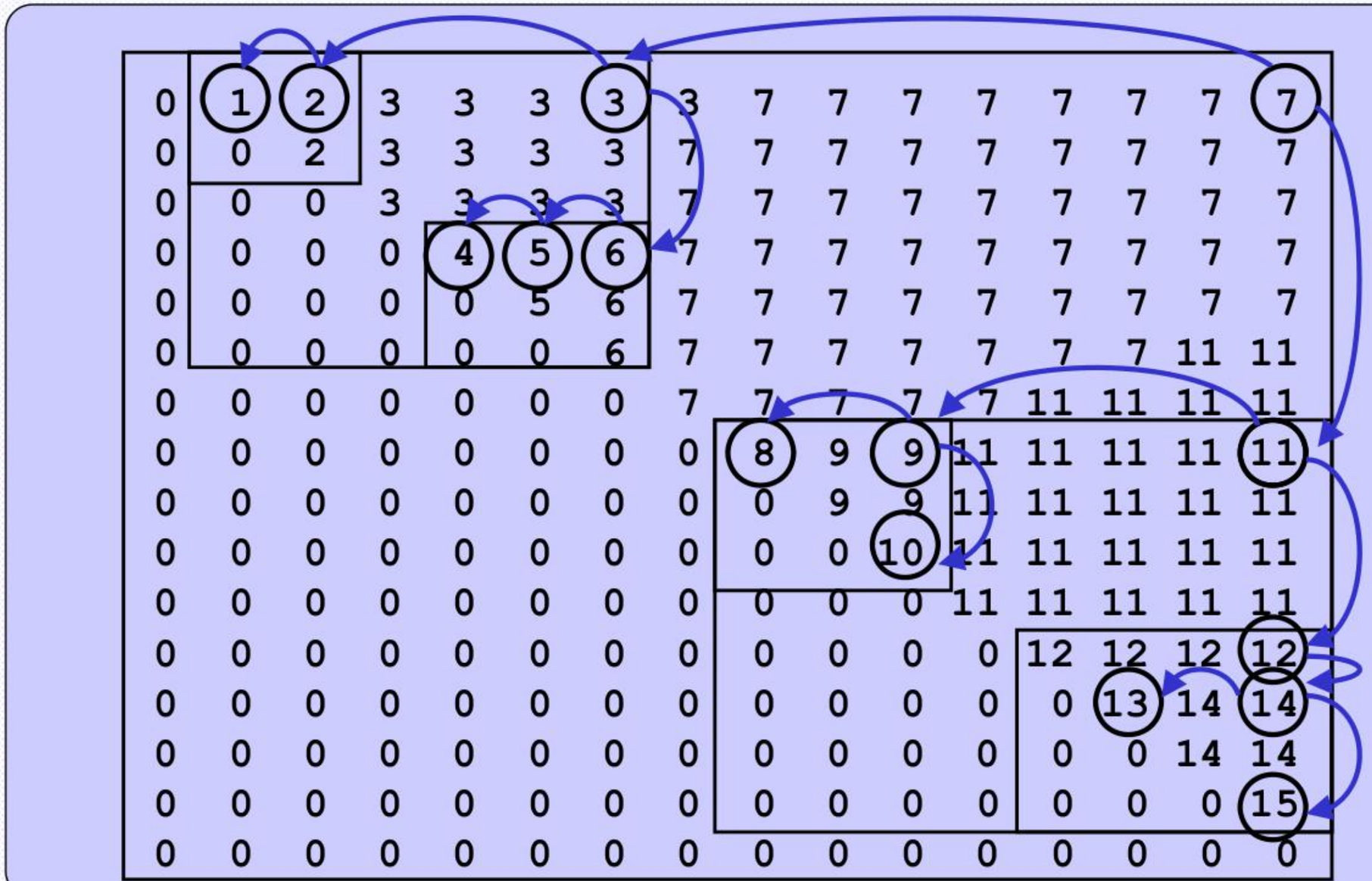






Výpočet optimálního BVS

Kořeny optimálních podstromů



Výpočet optimálního BVS

Vybudování optimálního stromu pomocí rekonstrukční tabulky kořenů

```
void buildTree( int L, int R) {

  if (R < L) return;

  int keyIndex = roots[L][R];
  // keys ... sorted array of keys
  int key = keys[roots[L][R]];

  insert(root, key);     // standard BST insert
  buildTree( L, keyIndex -1 );
  buildTree( keyIndex +1, R );
}
```

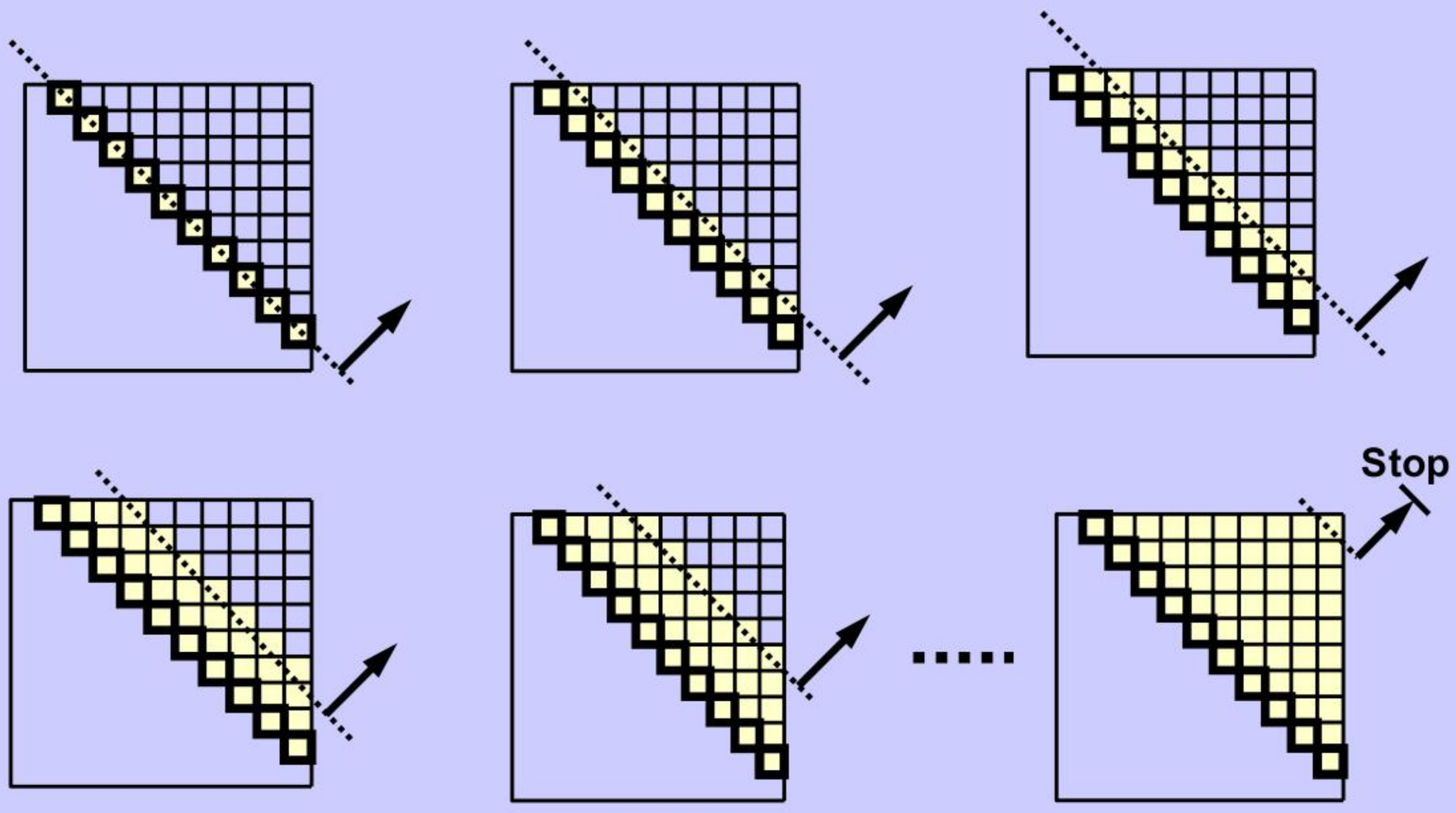
Výpočet optimálního BVS

Výpočet DP tabulek cen a kořenů

```
void optimalTree() {
  int L, R; double min;

  // size = 1
  for( i=0; i<=N; i++ ) {
    C[i][i] = pravděpodobnost[i]; roots[i][i] = i;

  // size > 1
  for( int size = 2; size <= N; size++ ) {
    L = 1; R = size;
    while( R <= N ) {
      C[L][R] = min(C[L][k-1]+C[k+1][R], k = L..R);
      roots[L][R] = 'k minimalizující předch. řádek';
      C[L][R] += sum(C[i][i], i = L..R);
      L++; R++;
} } }
```

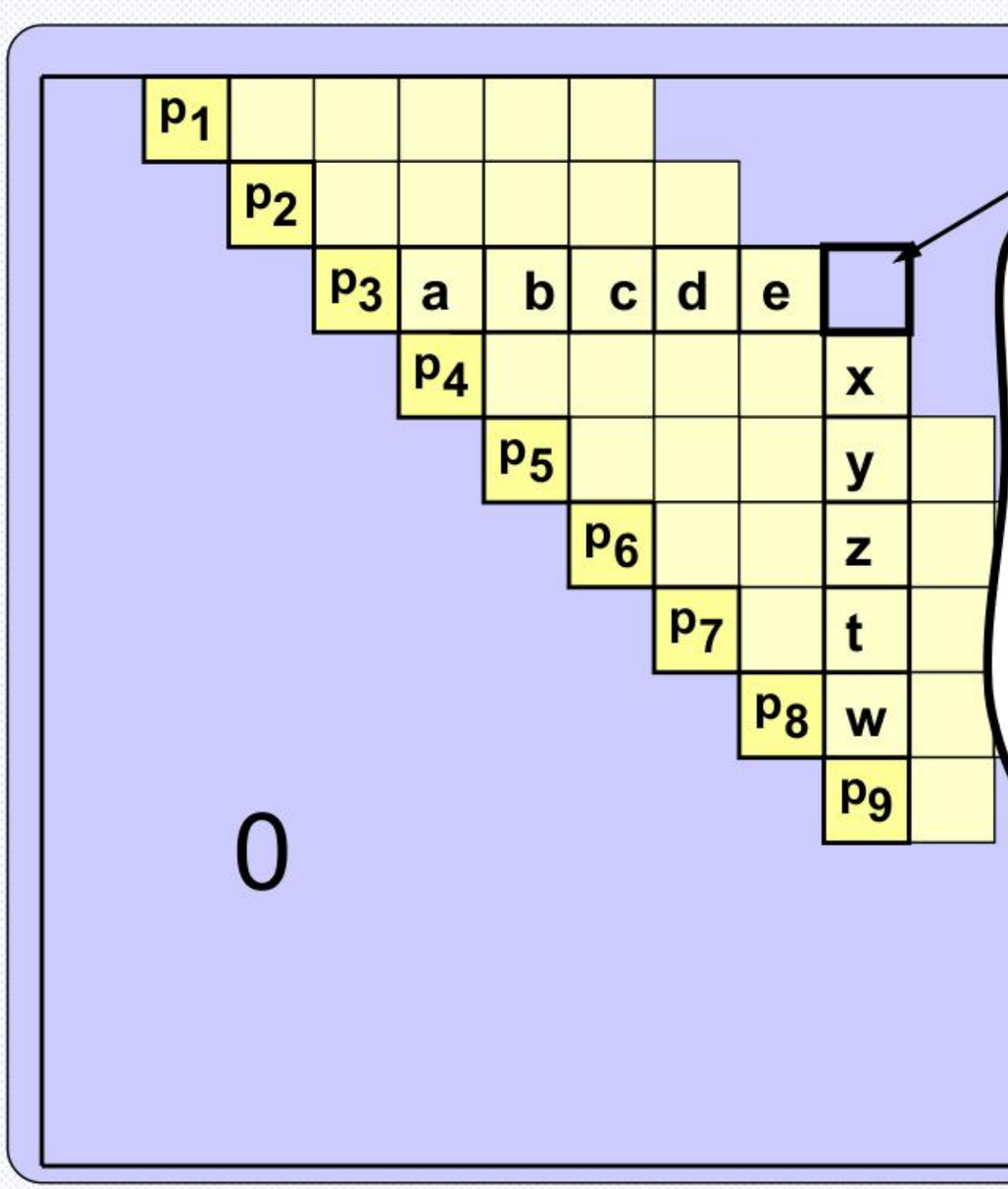
Výpočet optimálního BVS

Strategie DP
– nejprve se zpracují nejmenší podstromy, pak větší, atd…



Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



L=3, R=9

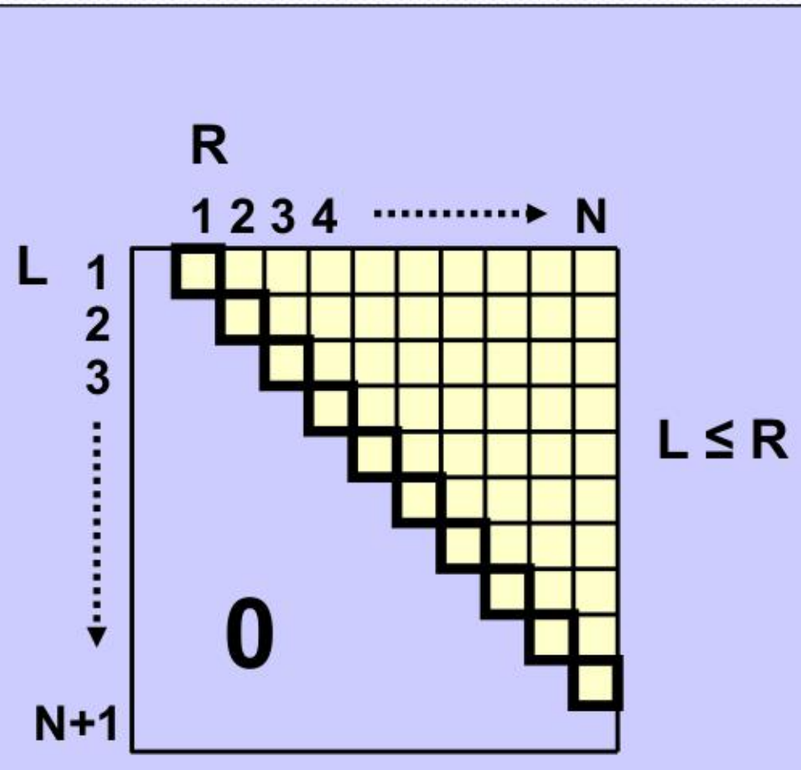
$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Datové struktury pro výpočet optmálního BVS

Ceny optimálních podstromů

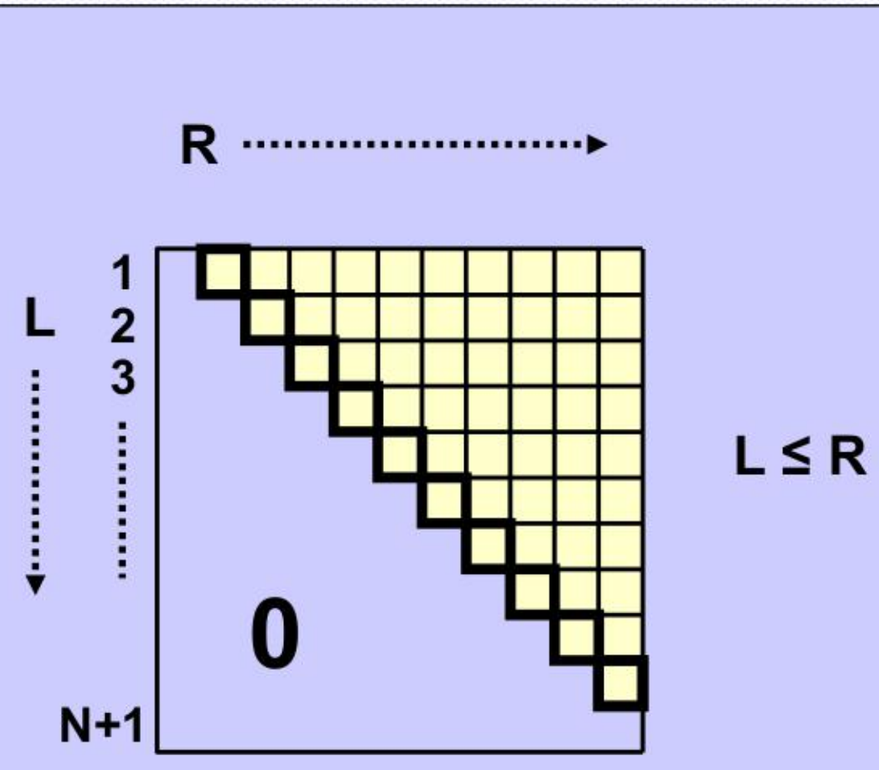
pole C [L][R] (L ≤ R)



diagonála ... L= R

Kořeny optimálních podstromů

pole roots [L][R] (L ≤ R)



diagonála ... L= R

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

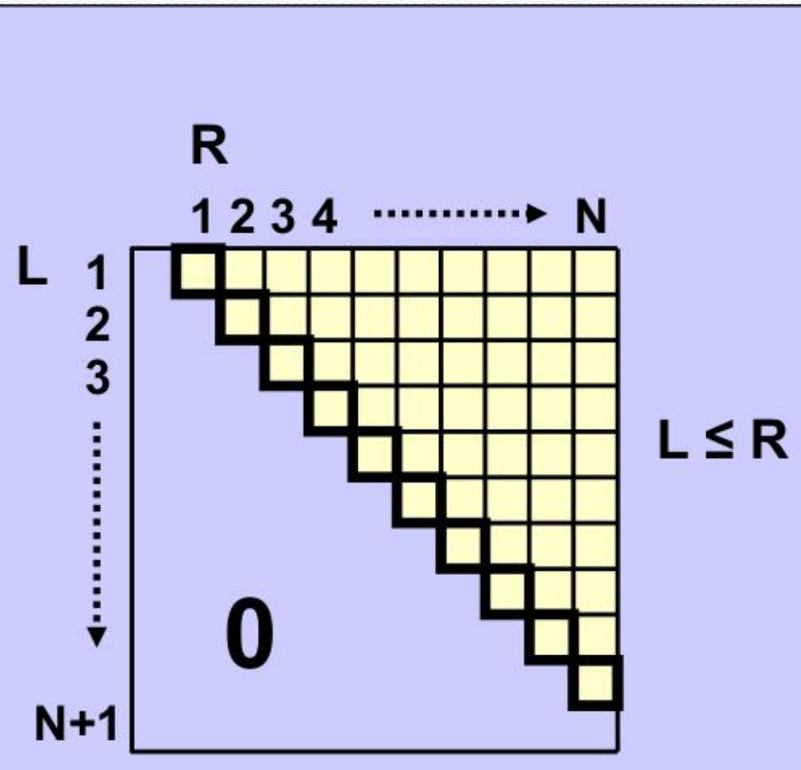
$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Datové struktury pro výpočet optmálního BVS

Ceny optimálních podstromů

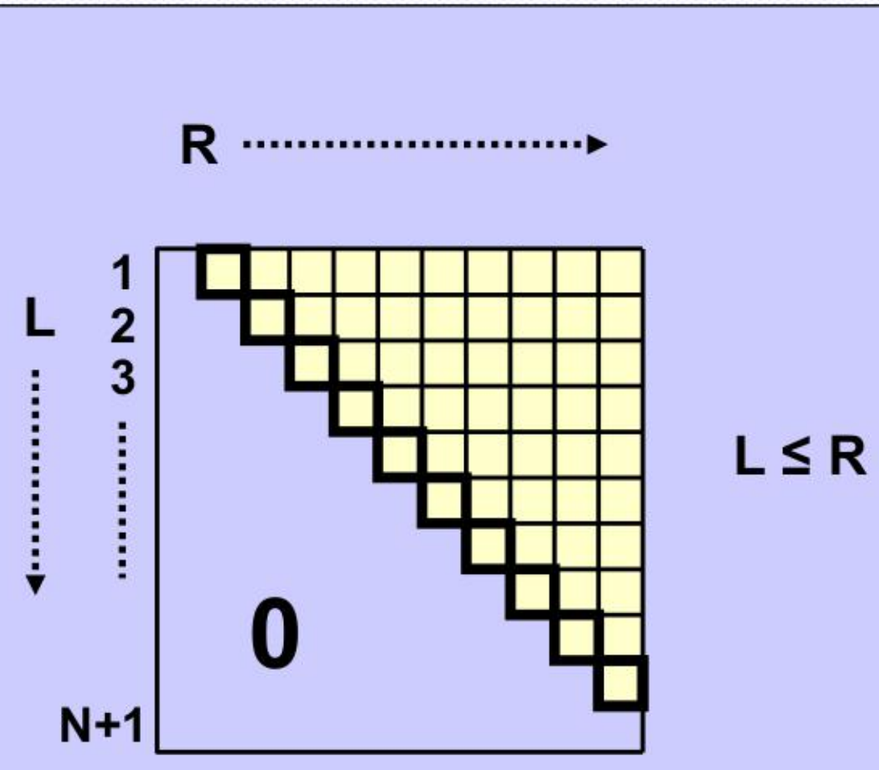
pole C [L][R] (L ≤ R)



diagonála ... L= R

Kořeny optimálních podstromů

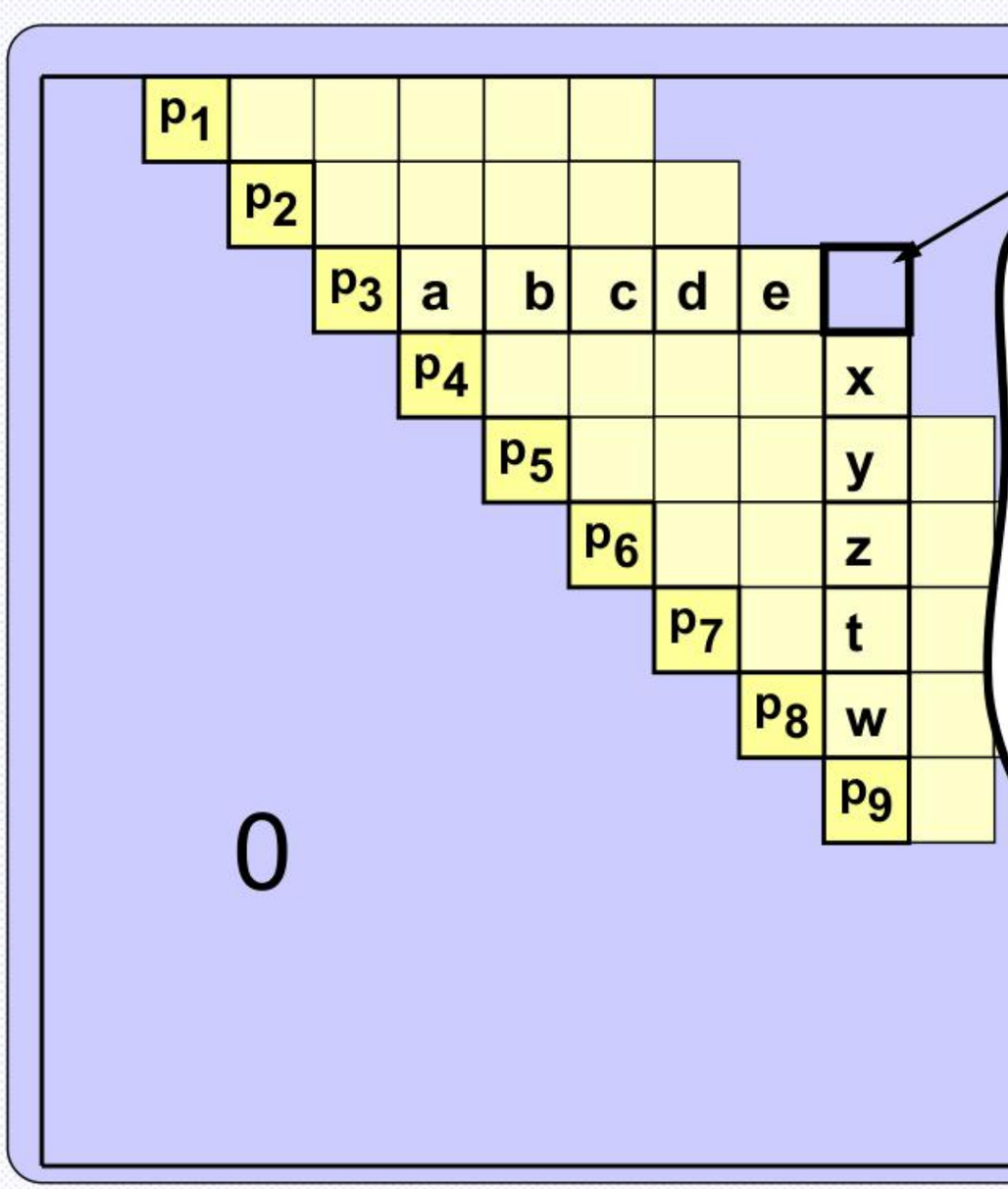
pole roots [L][R] (L ≤ R)



diagonála ... L= R

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



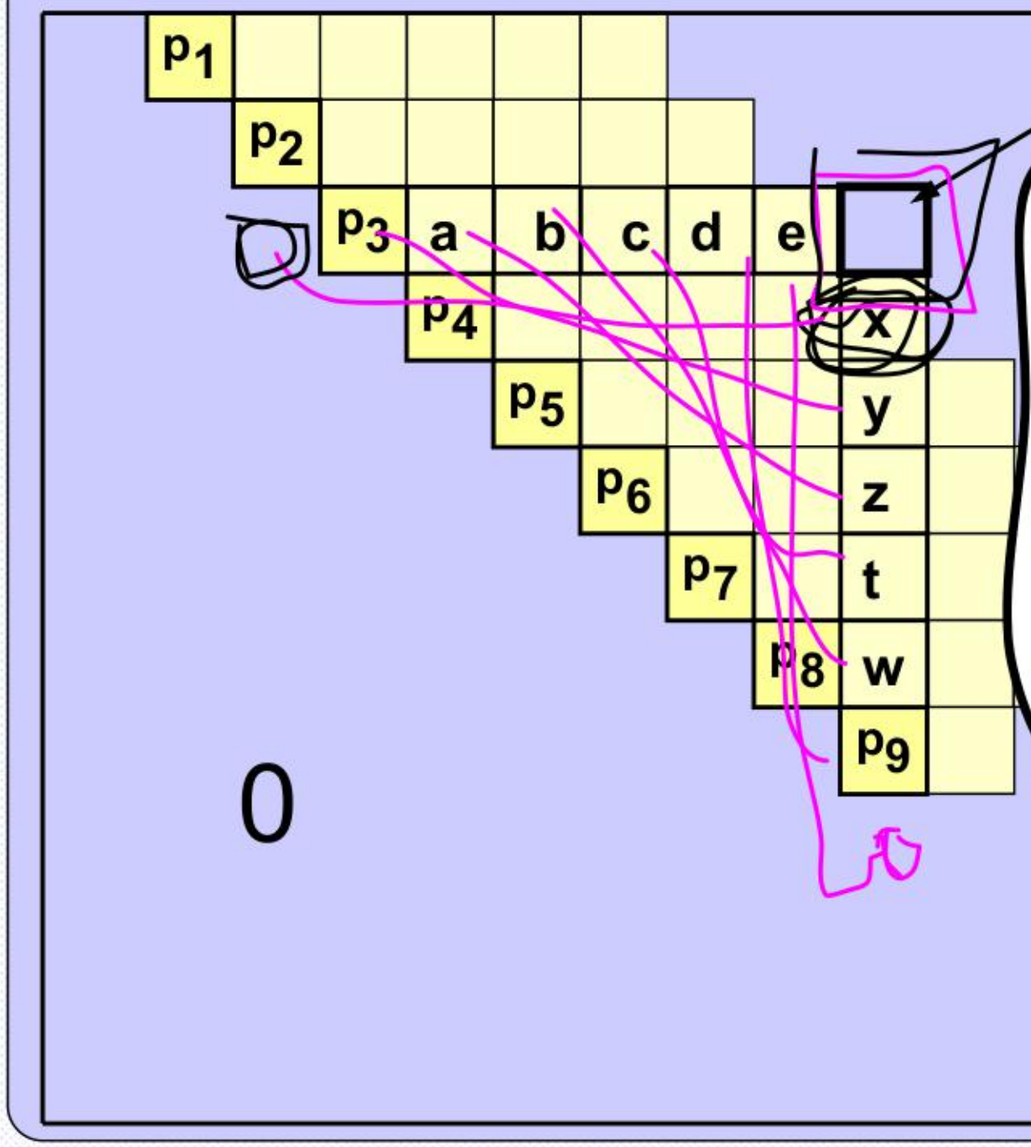
L=3, R=9

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



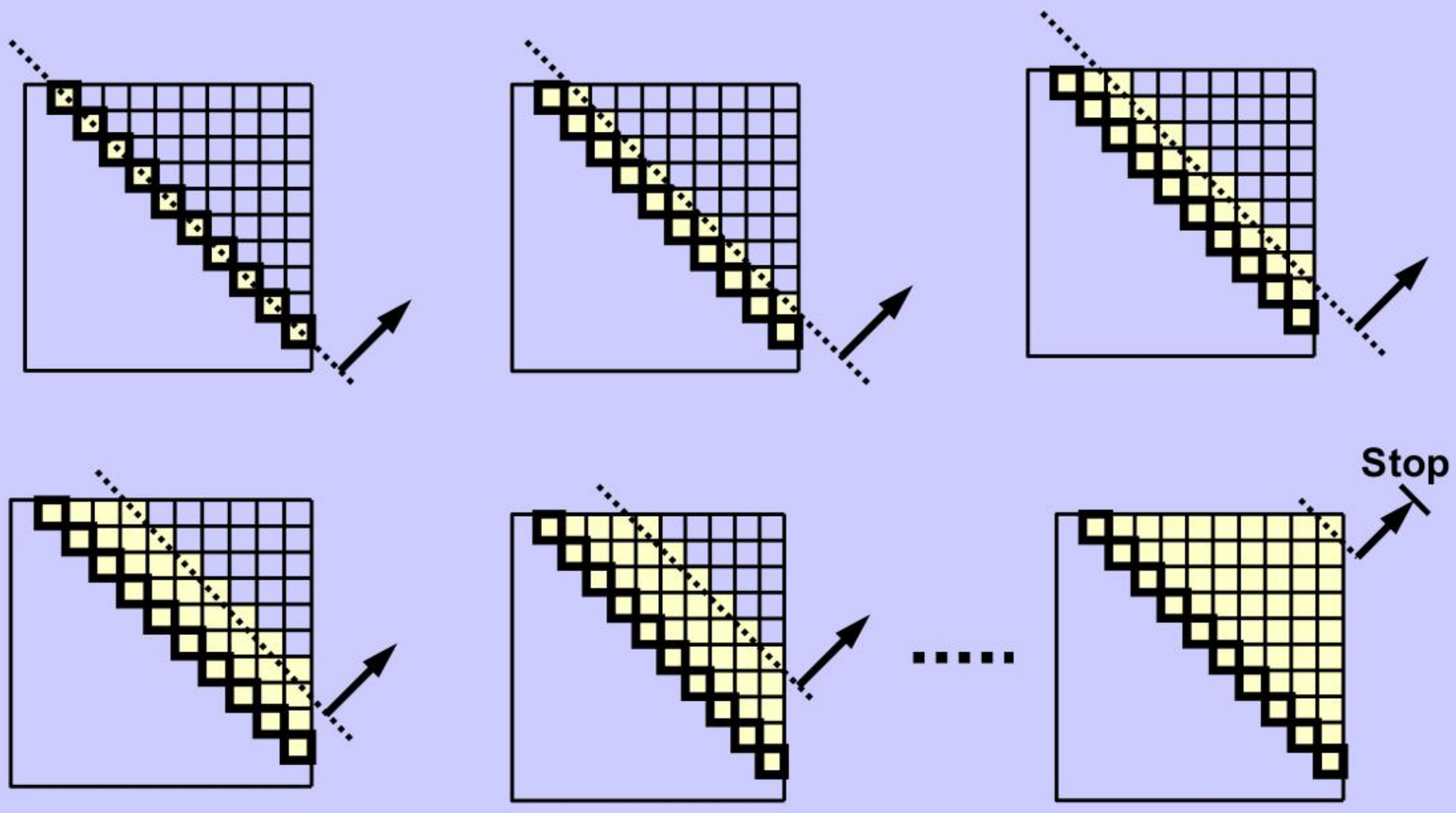
L=3, R=9

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Výpočet optimálního BVS

Strategie DP
– nejprve se zpracují nejmenší podstromy, pak větší, atd…



Výpočet optimálního BVS

Výpočet DP tabulek cen a kořenů

```
void optimalTree() {
  int L, R; double min;

  // size = 1
  for( i=0; i<=N; i++ ) {
    C[i][i] = pravděpodobnost[i]; roots[i][i] = i;

  // size > 1
  for( int size = 2; size <= N; size++ ) {
    L = 1; R = size;
    while( R <= N ) {
      C[L][R] = min(C[L][k-1]+C[k+1][R], k = L..R);
      roots[L][R] = 'k minimalizující předch. řádek';
      C[L][R] += sum(C[i][i], i = L..R);
      L++; R++;
} } }
```

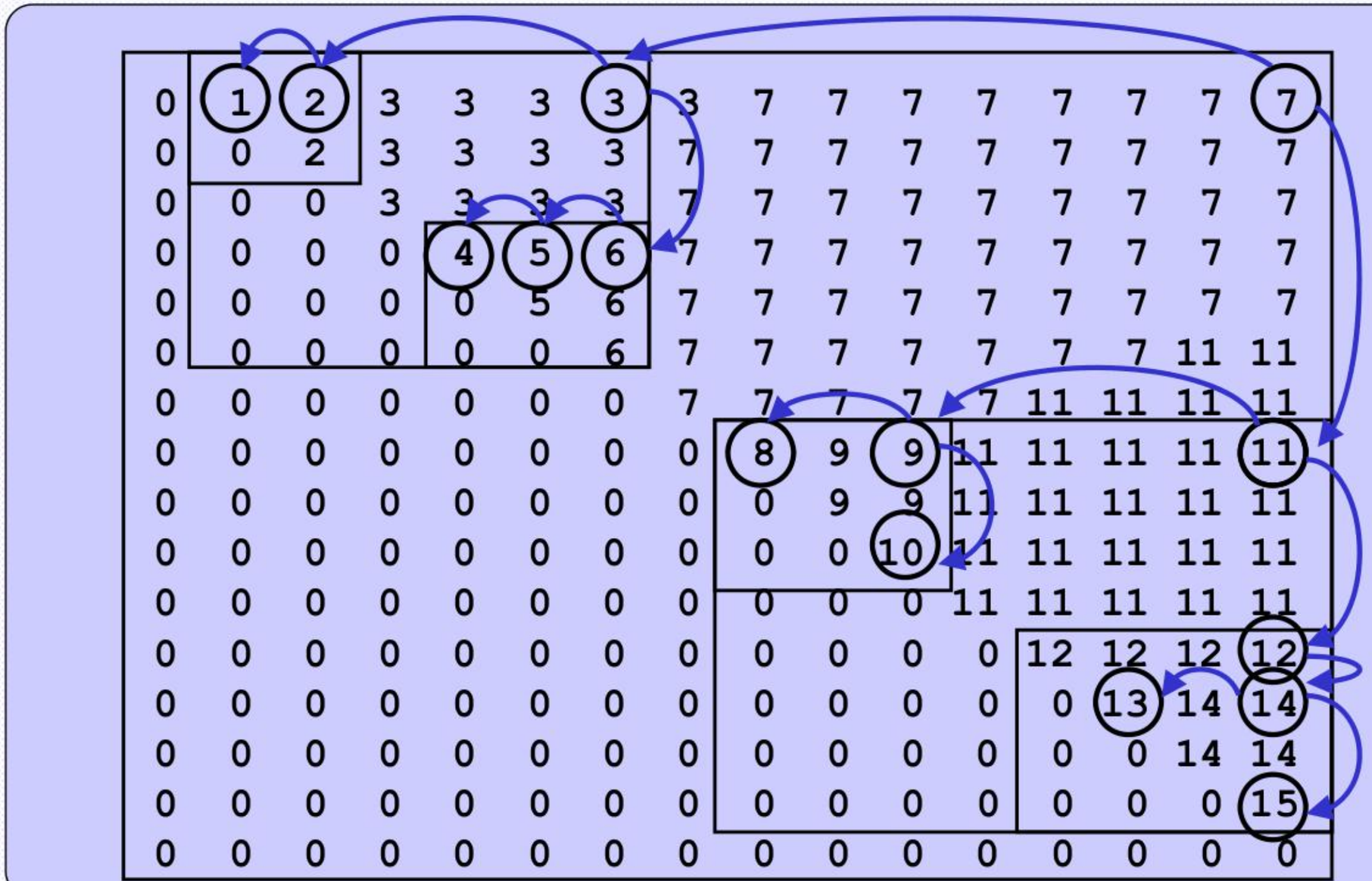
Výpočet optimálního BVS

Vybudování optimálního stromu pomocí rekonstrukční tabulky kořenů

```
void buildTree( int L, int R) {

  if (R < L) return;

  int keyIndex = roots[L][R];
  // keys ... sorted array of keys
  int key = keys[roots[L][R]];

  insert(root, key);     // standard BST insert
  buildTree( L, keyIndex -1 );
  buildTree( keyIndex +1, R );
}
```

Výpočet optimálního BVS

Kořeny optimálních podstromů



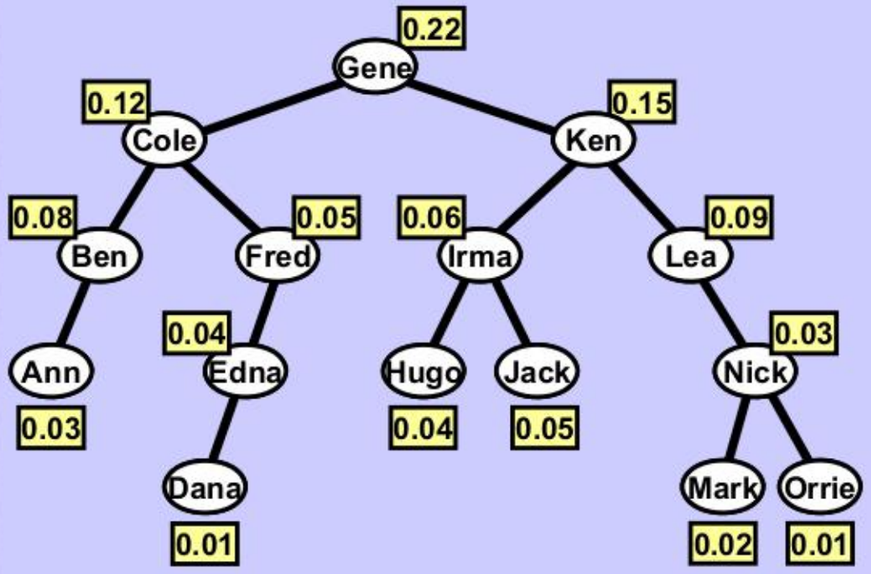
Výpočet optimálního BVS

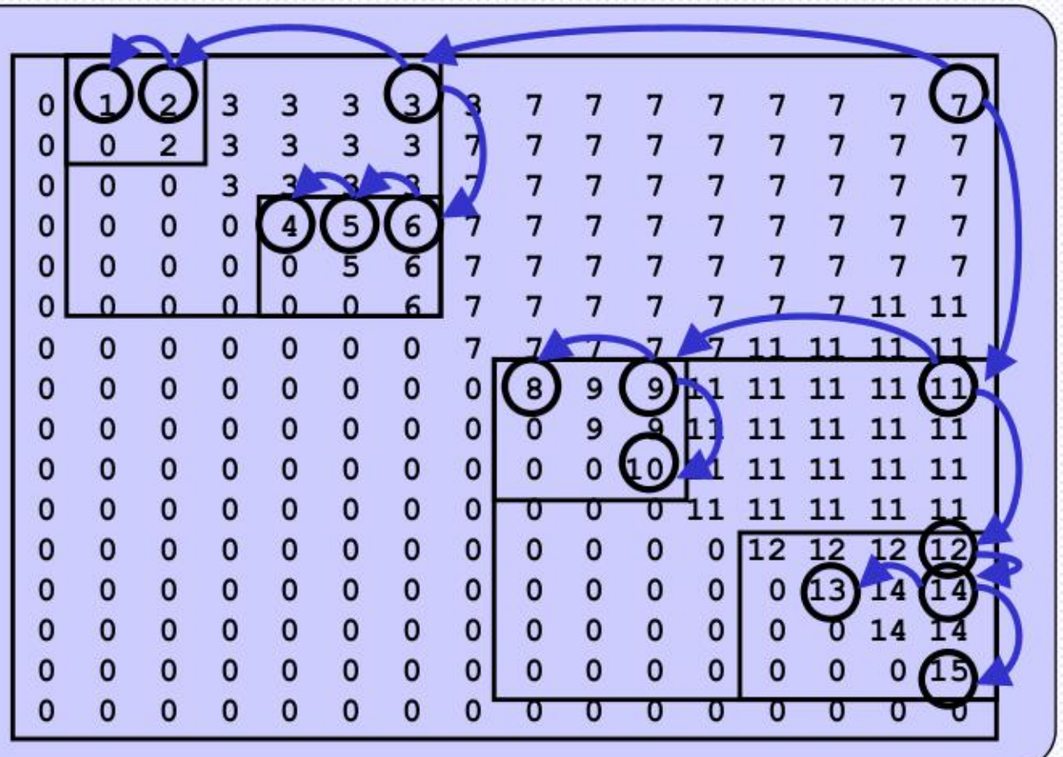
Kořeny optimálních podstromů

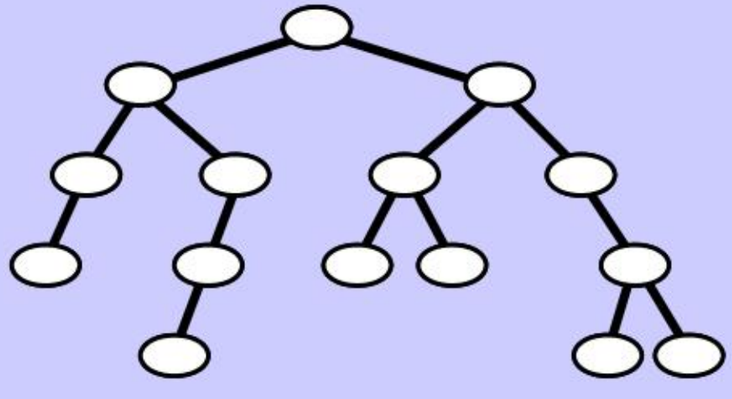
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	3	3	3	3	7	7	7	7	7	7	7	7
0	0	2	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	0	4	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	0	6	7	7	7	7	7	7	7	11	11
0	0	0	0	0	0	0	7	7	7	7	7	11	11	11	11
0	0	0	0	0	0	0	0	8	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	10	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	0	12	12	12	12
0	0	0	0	0	0	0	0	0	0	0	0	0	13	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

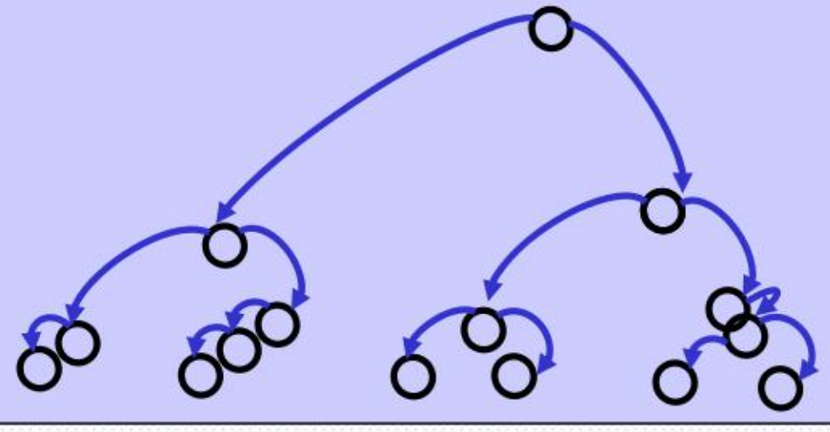
Výpočet optimálního BVS

Korespondence stromů









Výpočet optimálního BVS

Ceny optimálních podstromů

	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

Výpočet optimálního BVS

Ceny optimálních podstromů

	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

Dynamické programování

Nejdelší společná podposloupnost

Výpočet optimálního BVS

Ceny optimálních podstromů

	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

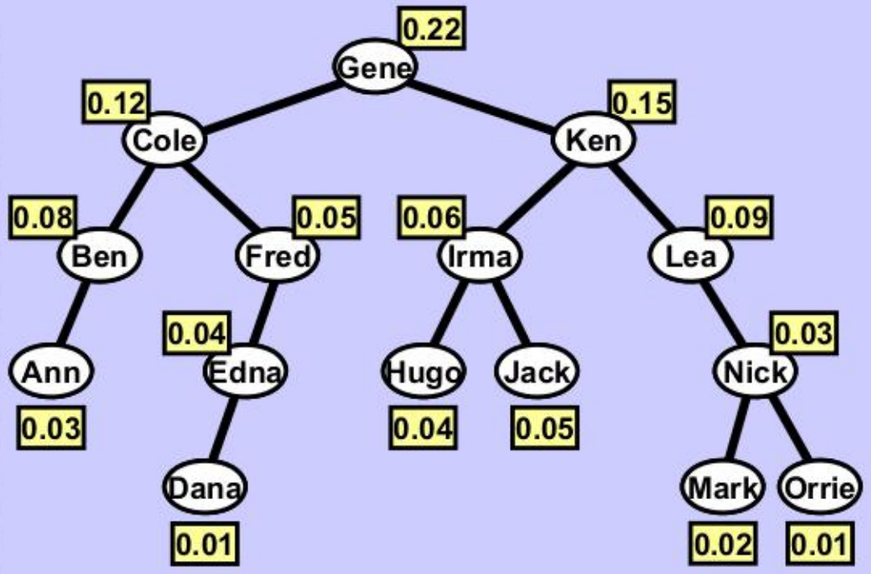
Výpočet optimálního BVS

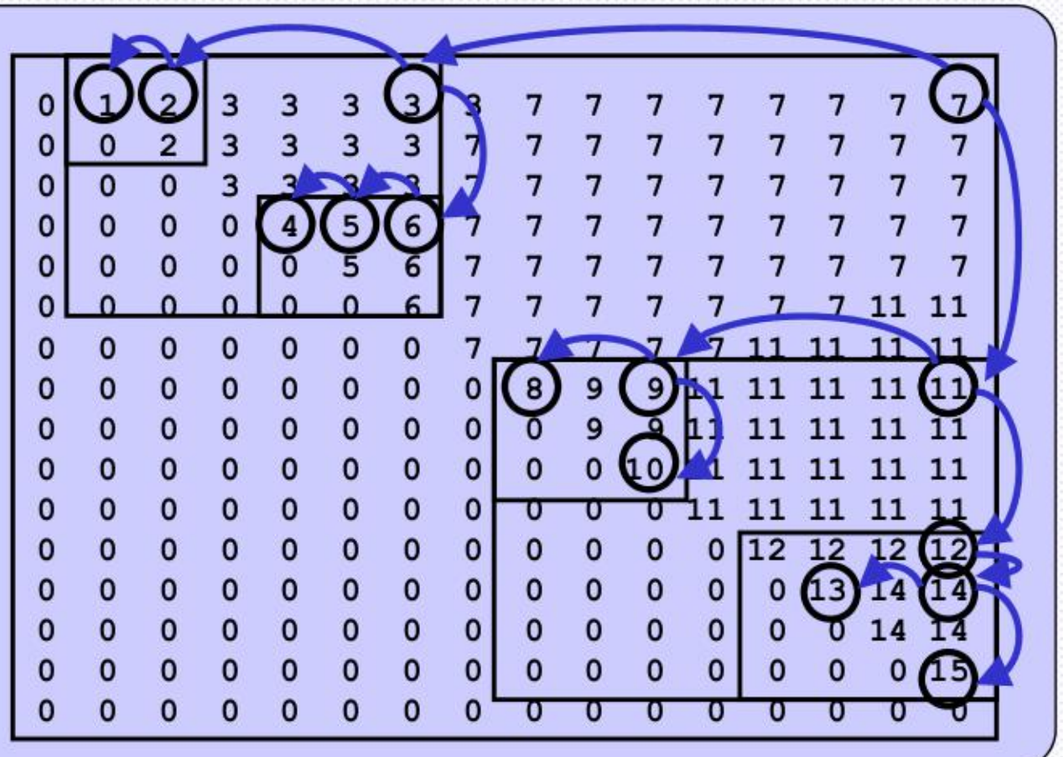
Ceny optimálních podstromů

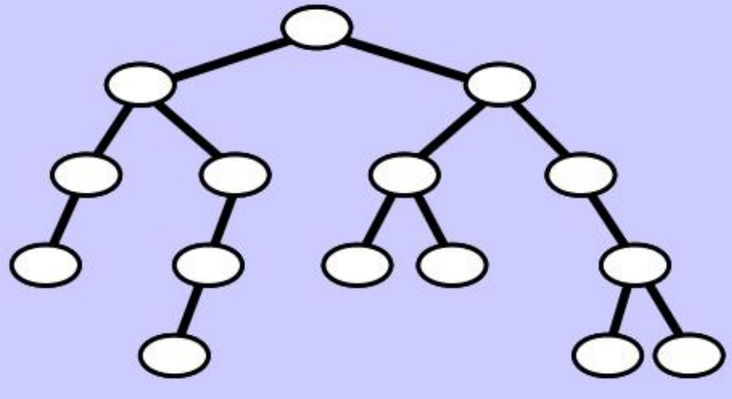
	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

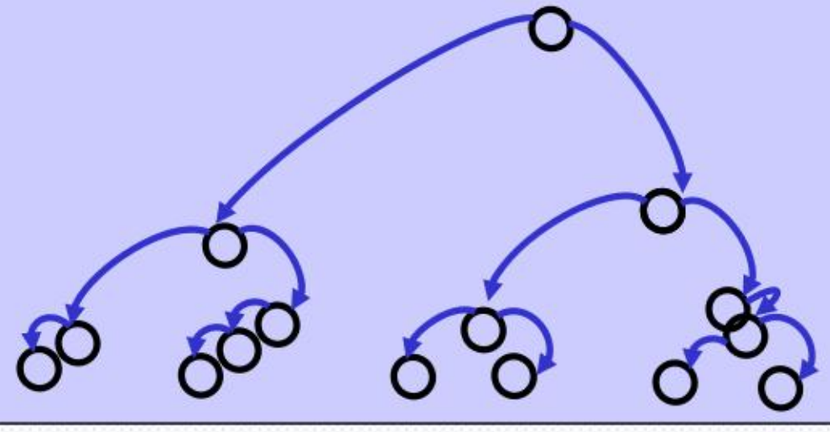
Výpočet optimálního BVS

Korespondence stromů









Výpočet optimálního BVS

Kořeny optimálních podstromů

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	3	3	3	3	7	7	7	7	7	7	7	7
0	0	2	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	0	4	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	0	6	7	7	7	7	7	7	7	11	11
0	0	0	0	0	0	0	7	7	7	7	7	11	11	11	11
0	0	0	0	0	0	0	0	8	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	10	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	0	12	12	12	12
0	0	0	0	0	0	0	0	0	0	0	0	0	13	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Výpočet optimálního BVS

Kořeny optimálních podstromů

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	3	3	3	3	7	7	7	7	7	7	7	7
0	0	2	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	0	4	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	0	6	7	7	7	7	7	7	7	11	11
0	0	0	0	0	0	0	7	7	7	7	7	11	11	11	11
0	0	0	0	0	0	0	0	8	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	10	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	0	12	12	12	12
0	0	0	0	0	0	0	0	0	0	0	0	0	13	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Výpočet optimálního BVS

Vybudování optimálního stromu pomocí rekonstrukční tabulky kořenů

```
void buildTree( int L, int R) {

  if (R < L) return;

  int keyIndex = roots[L][R];
  // keys ... sorted array of keys
  int key = keys[roots[L][R]];

  insert(root, key);     // standard BST insert
  buildTree( L, keyIndex -1 );
  buildTree( keyIndex +1, R );
}
```

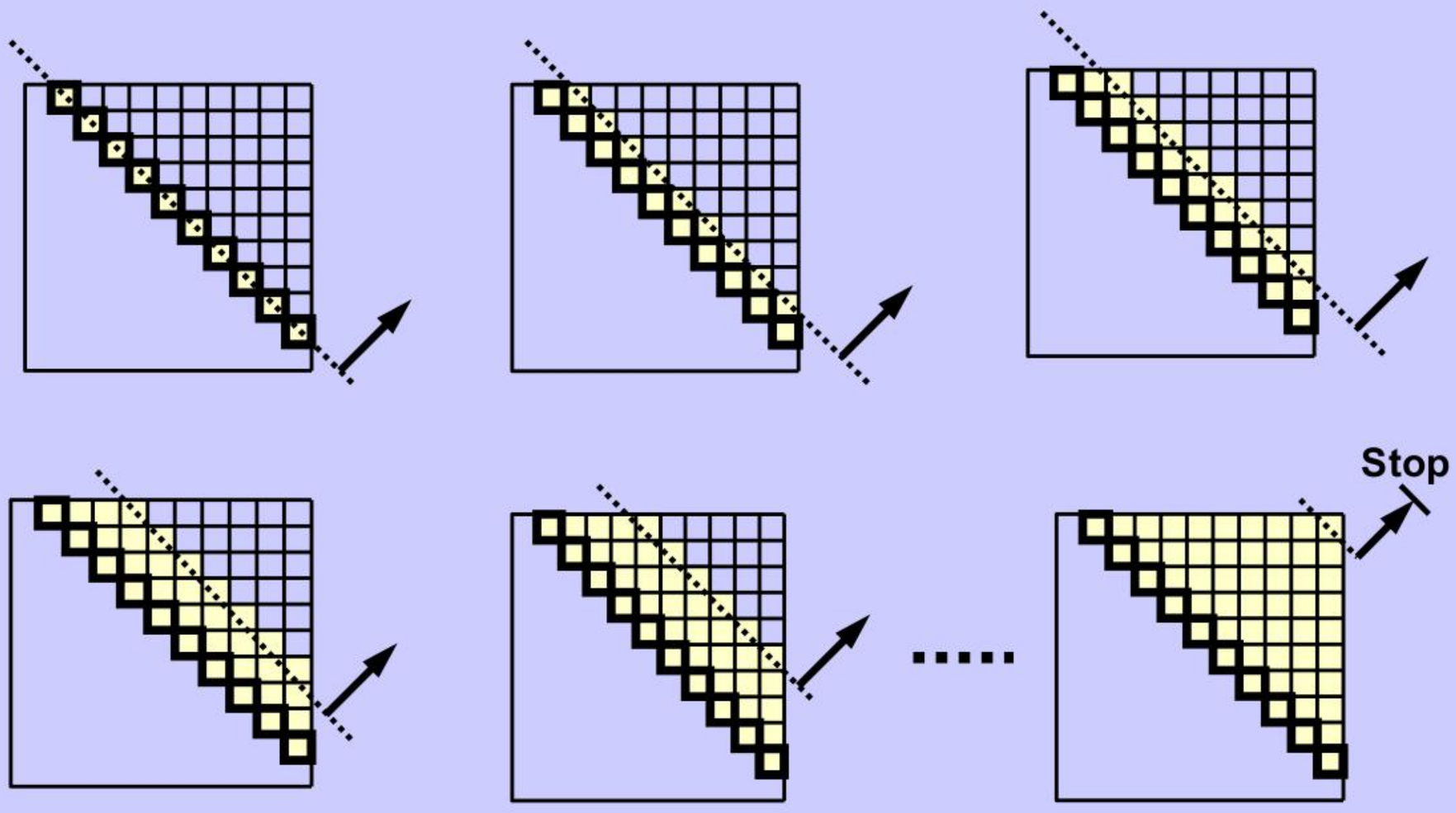
Výpočet optimálního BVS

Výpočet DP tabulek cen a kořenů

```
void optimalTree() {
  int L, R; double min;

  // size = 1
  for( i=0; i<=N; i++ ) {
    C[i][i] = pravděpodobnost[i]; roots[i][i] = i;

  // size > 1
  for( int size = 2; size <= N; size++ ) {
    L = 1; R = size;
    while( R <= N ) {
      C[L][R] = min(C[L][k-1]+C[k+1][R], k = L..R);
      roots[L][R] = 'k minimalizující předch. řádek';
      C[L][R] += sum(C[i][i], i = L..R);
      L++; R++;
} } }
```

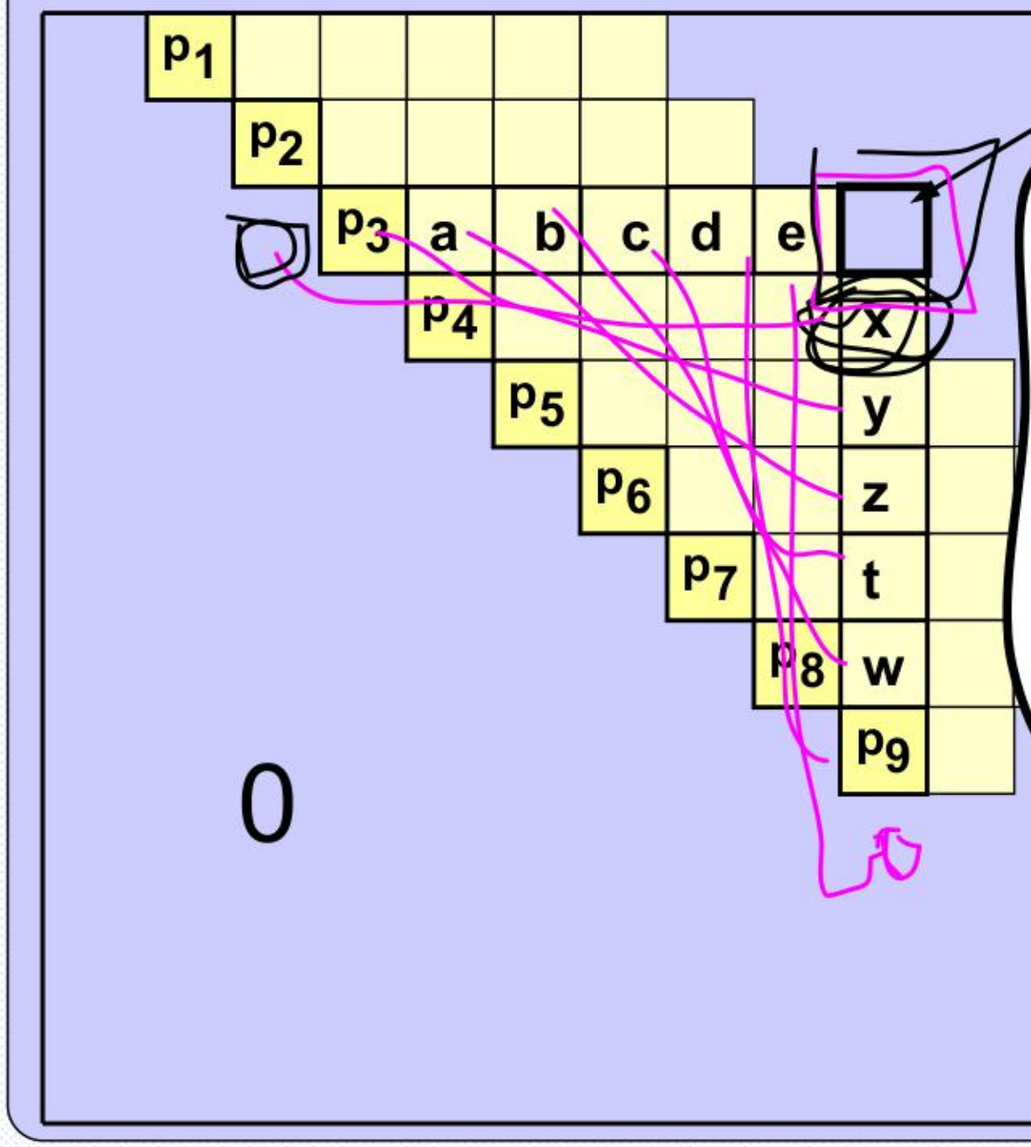
Výpočet optimálního BVS

Strategie DP
– nejprve se zpracují nejmenší podstromy, pak větší, atd…



Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



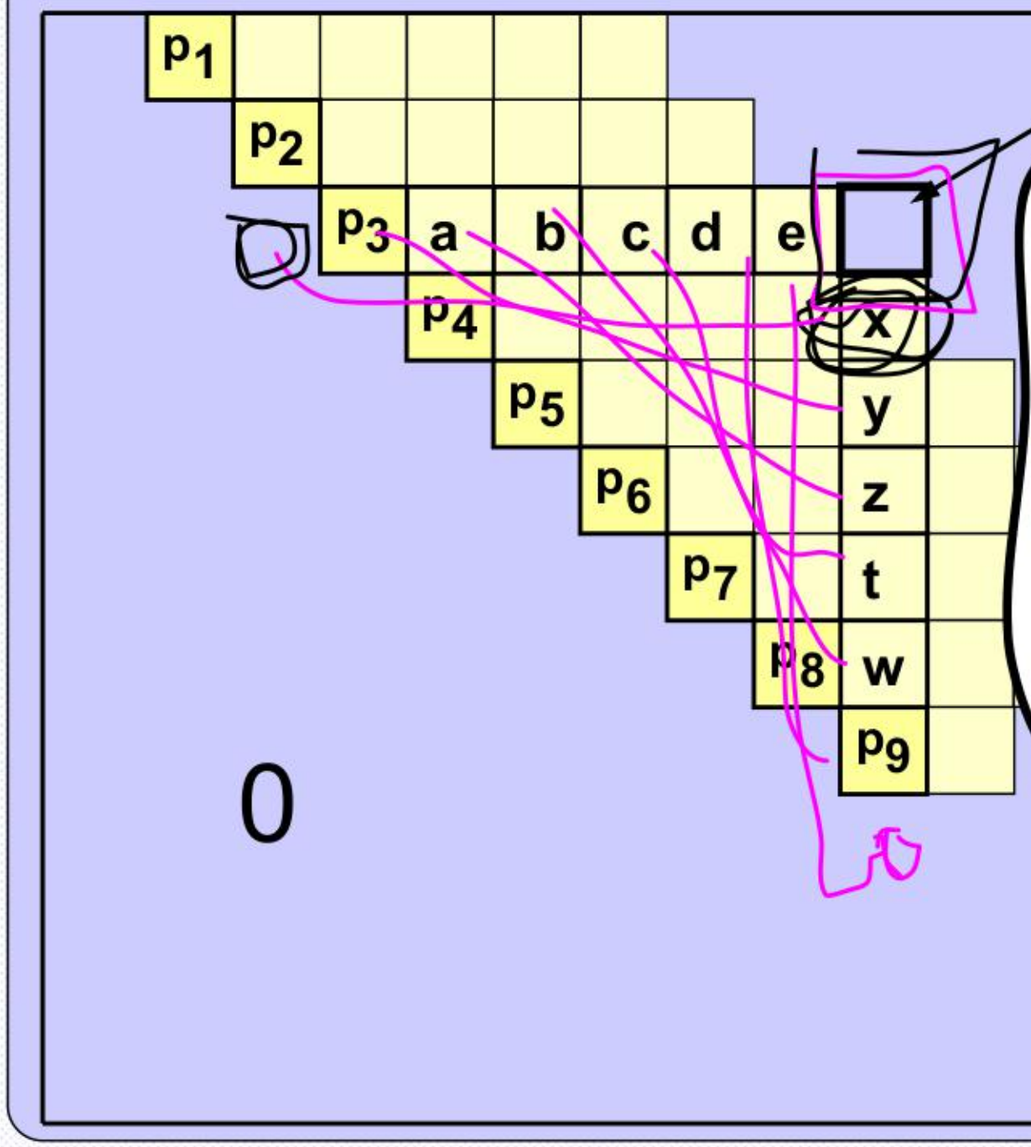
L=3, R=9

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



L=3, R=9

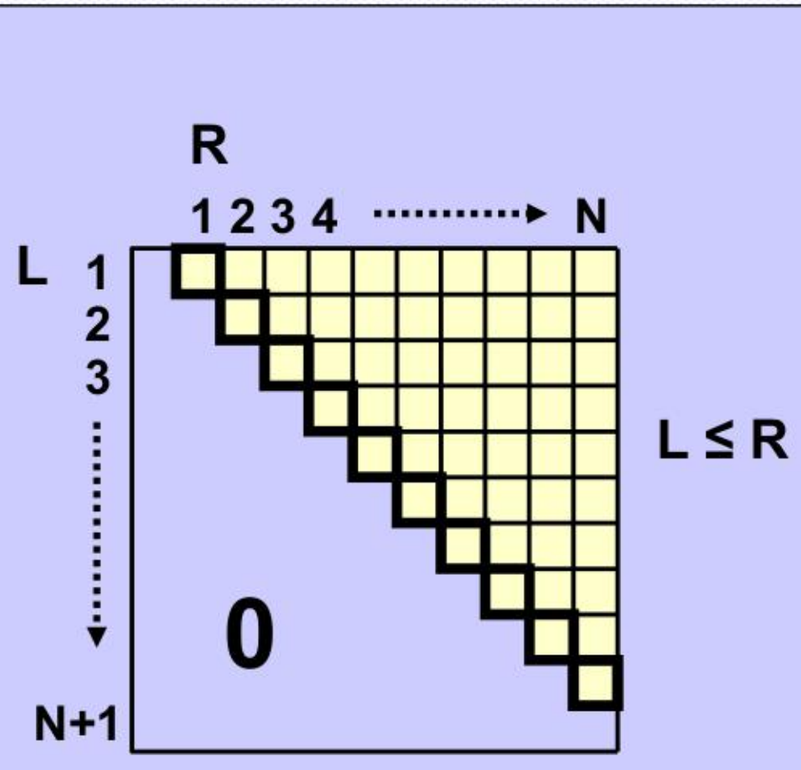
$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Datové struktury pro výpočet optmálního BVS

Ceny optimálních podstromů

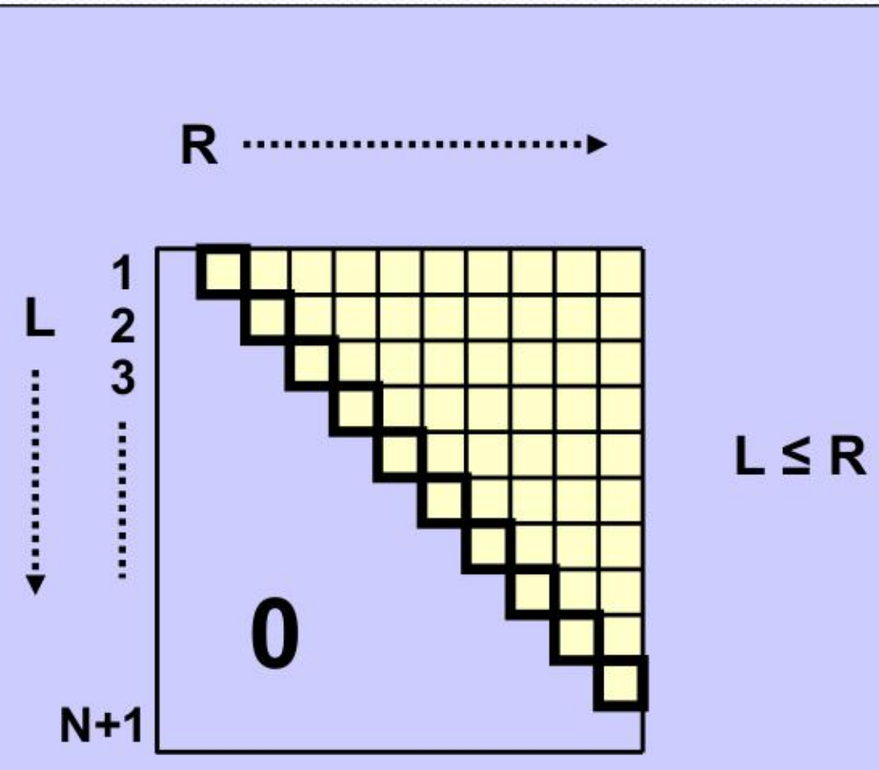
pole C [L][R] (L ≤ R)



diagonála ... L= R

Kořeny optimálních podstromů

pole roots [L][R] (L ≤ R)



diagonála ... L= R

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

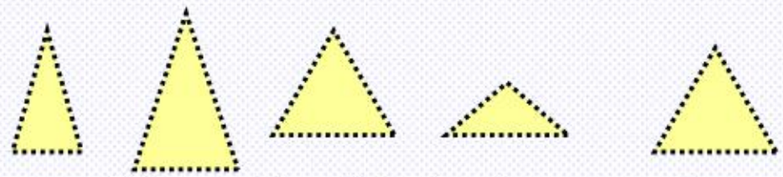
$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

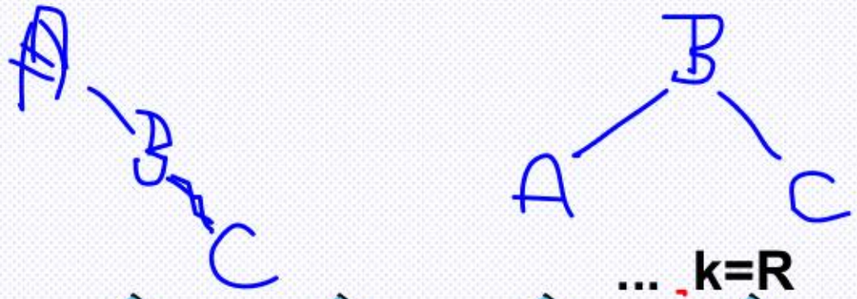
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



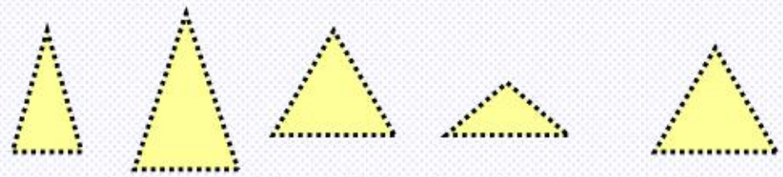
3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

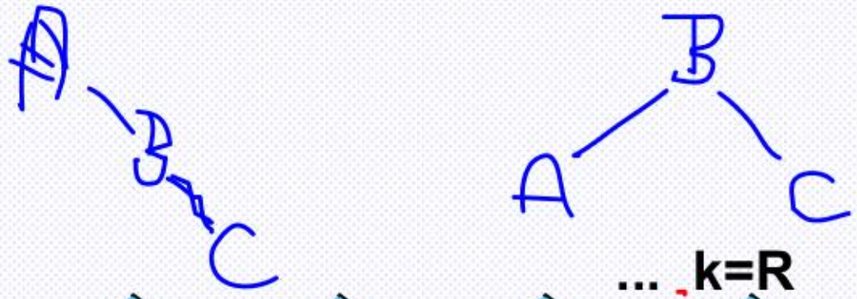
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: k = L, L+1, L+2, ..., R



3. Zaregistruj index k, který minimalizuje cenu, tj. hodnotu

$$c_{1k} + \sum_{i=L}^{k-1} p_i + c_{2k} + \sum_{i=k+1}^{R} p_i + p_k$$

4. Klíč s indexem k je kořenem optimálního stromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Minimalizace ceny BVS

C(L,R) ..... Cena optimálního podstromu obsahujícího klíče s indexy L, L+1, L+2, ..., R-1, R

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^{R} p_i + p_k \} =$$

$$= \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^{R} p_i \} =$$

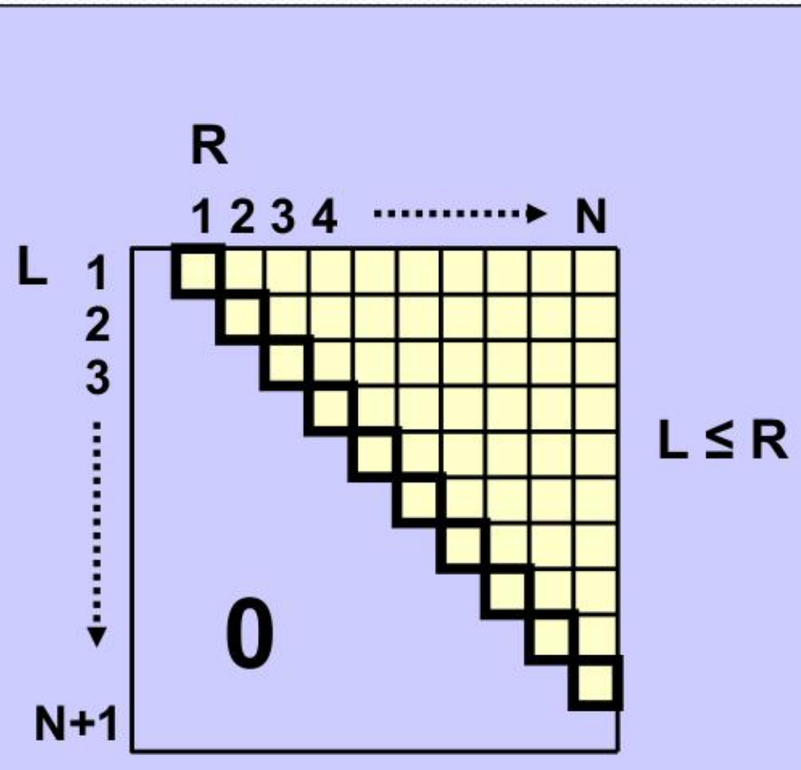
$$(*) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

Hodnota k minimalizující (*) je indexem kořenu optim. podstromu.

Datové struktury pro výpočet optmálního BVS

Ceny optimálních podstromů

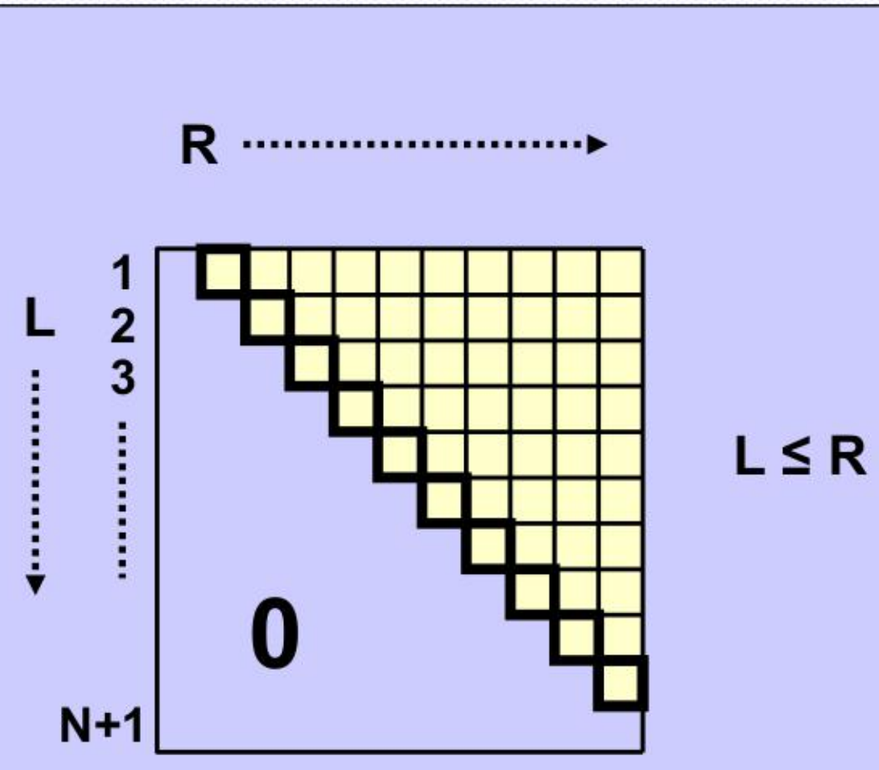
pole C [L][R] (L ≤ R)



diagonála ... L= R

Kořeny optimálních podstromů

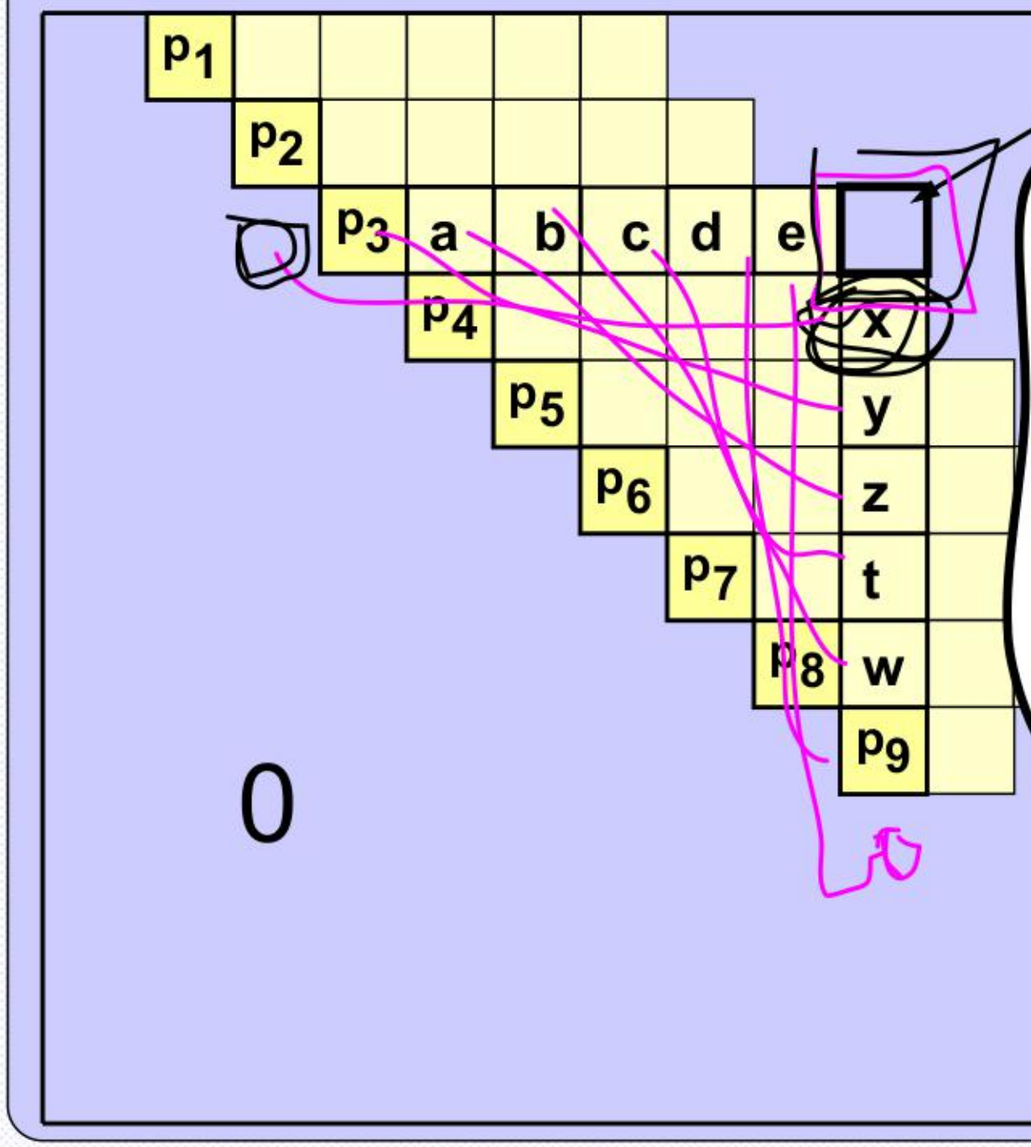
pole roots [L][R] (L ≤ R)



diagonála ... L= R

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



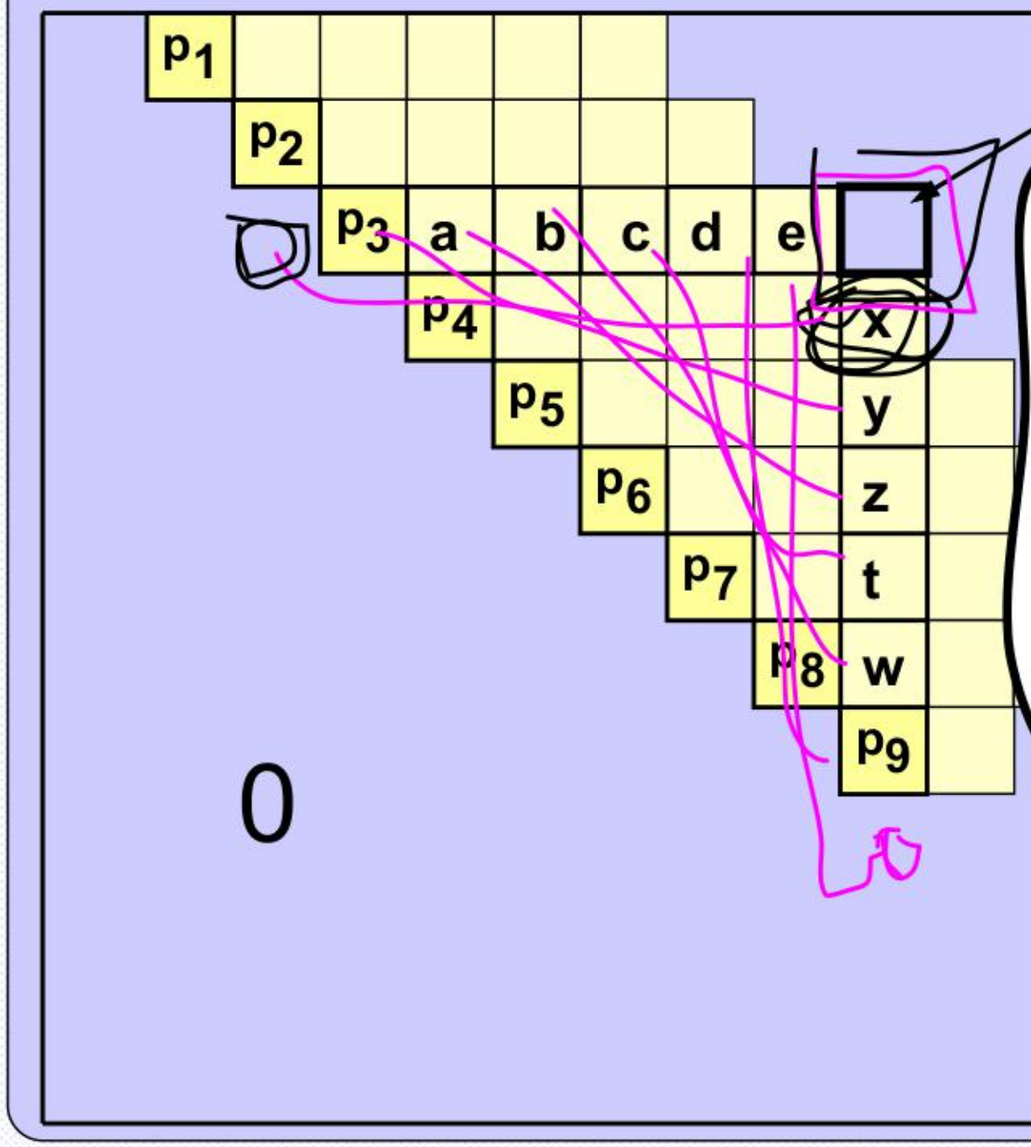
L=3, R=9

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



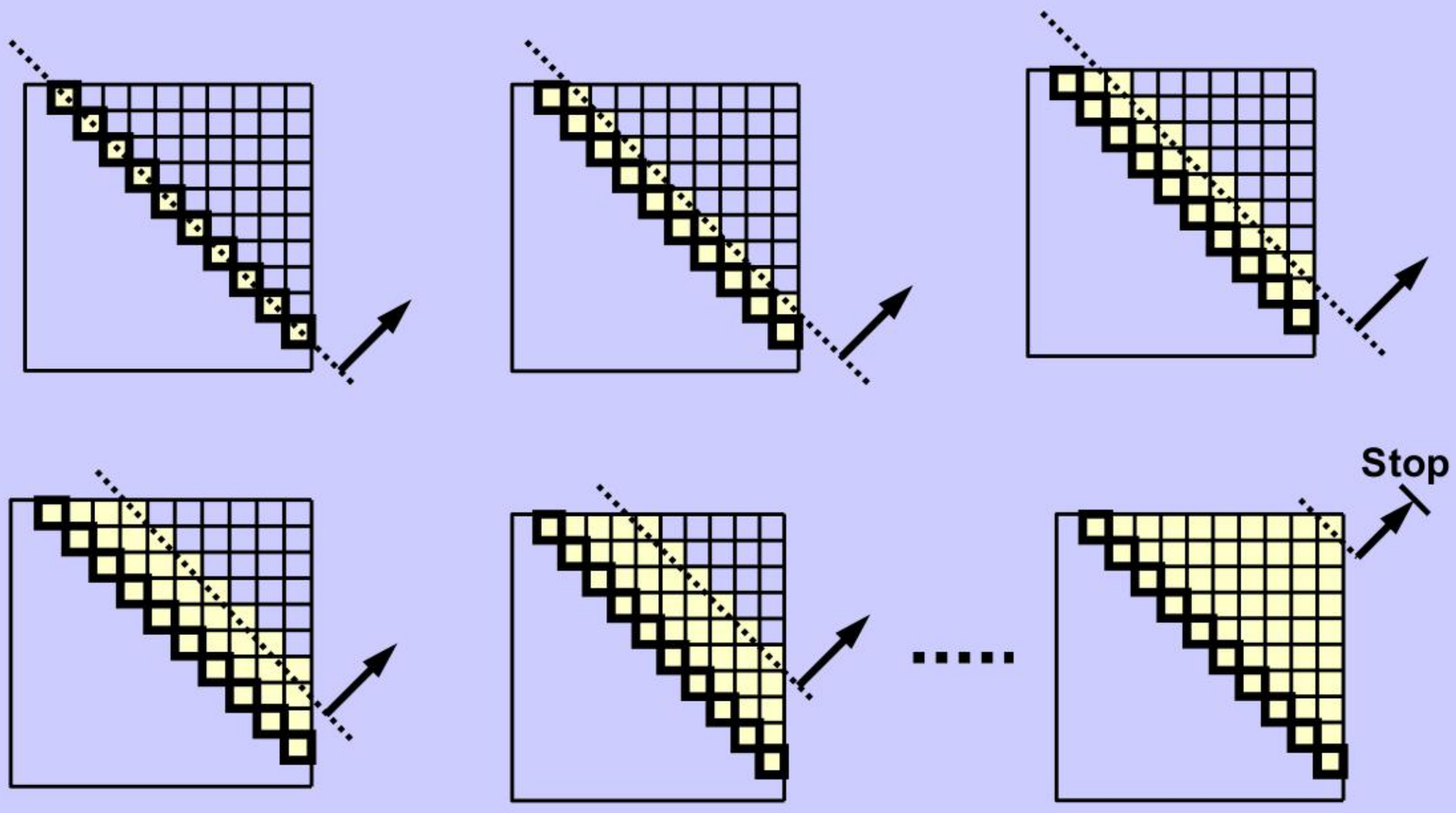
L=3, R=9

$$C(L,R) = \min_{L \le k \le R} \{ C(L, k-1) + C(k+1,R) \} + \sum_{i=L}^{R} p_i$$

$$C(L,R) = \min\{ 0+x,\ p_3+y,\ a+z,\ b+t,\ c+w,\ d+p_9,\ e+0 \} + \sum_{i=L}^{R} p_i$$

Výpočet optimálního BVS

Strategie DP
– nejprve se zpracují nejmenší podstromy, pak větší, atd…



Výpočet optimálního BVS

Výpočet DP tabulek cen a kořenů

```
void optimalTree() {
  int L, R; double min;

  // size = 1
  for( i=0; i<=N; i++ ) {
    C[i][i] = pravděpodobnost[i]; roots[i][i] = i;

  // size > 1
  for( int size = 2; size <= N; size++ ) {
    L = 1; R = size;
    while( R <= N ) {
      C[L][R] = min(C[L][k-1]+C[k+1][R], k = L..R);
      roots[L][R] = 'k minimalizující předch. řádek';
      C[L][R] += sum(C[i][i], i = L..R);
      L++; R++;
} } }
```

Výpočet optimálního BVS

Vybudování optimálního stromu pomocí rekonstrukční tabulky kořenů

```
void buildTree( int L, int R) {

  if (R < L) return;

  int keyIndex = roots[L][R];
  // keys ... sorted array of keys
  int key = keys[roots[L][R]];

  insert(root, key);     // standard BST insert
  buildTree( L, keyIndex -1 );
  buildTree( keyIndex +1, R );
}
```

Výpočet optimálního BVS

Kořeny optimálních podstromů

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	3	3	3	3	7	7	7	7	7	7	7	7
0	0	2	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	0	4	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	0	6	7	7	7	7	7	7	7	11	11
0	0	0	0	0	0	0	7	7	7	7	7	11	11	11	11
0	0	0	0	0	0	0	0	8	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	10	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	0	12	12	12	12
0	0	0	0	0	0	0	0	0	0	0	0	0	13	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

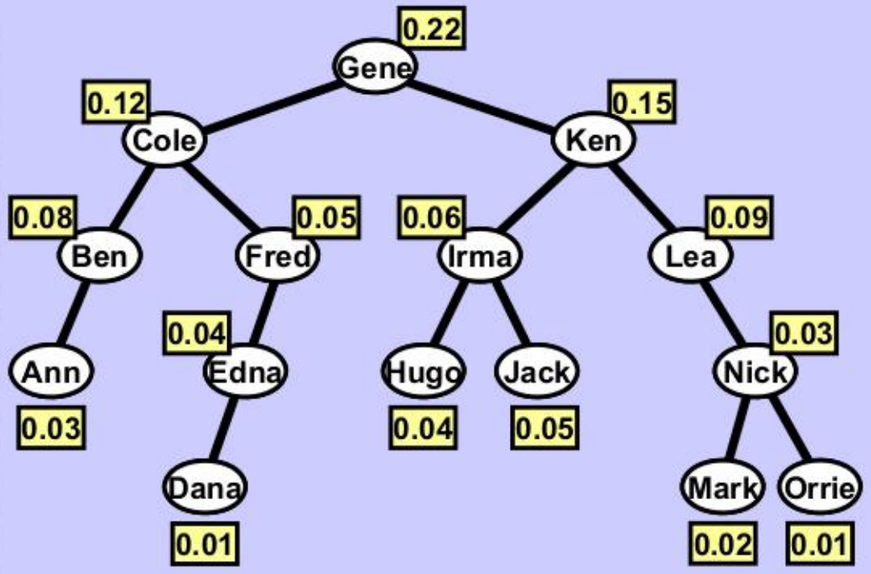
Výpočet optimálního BVS

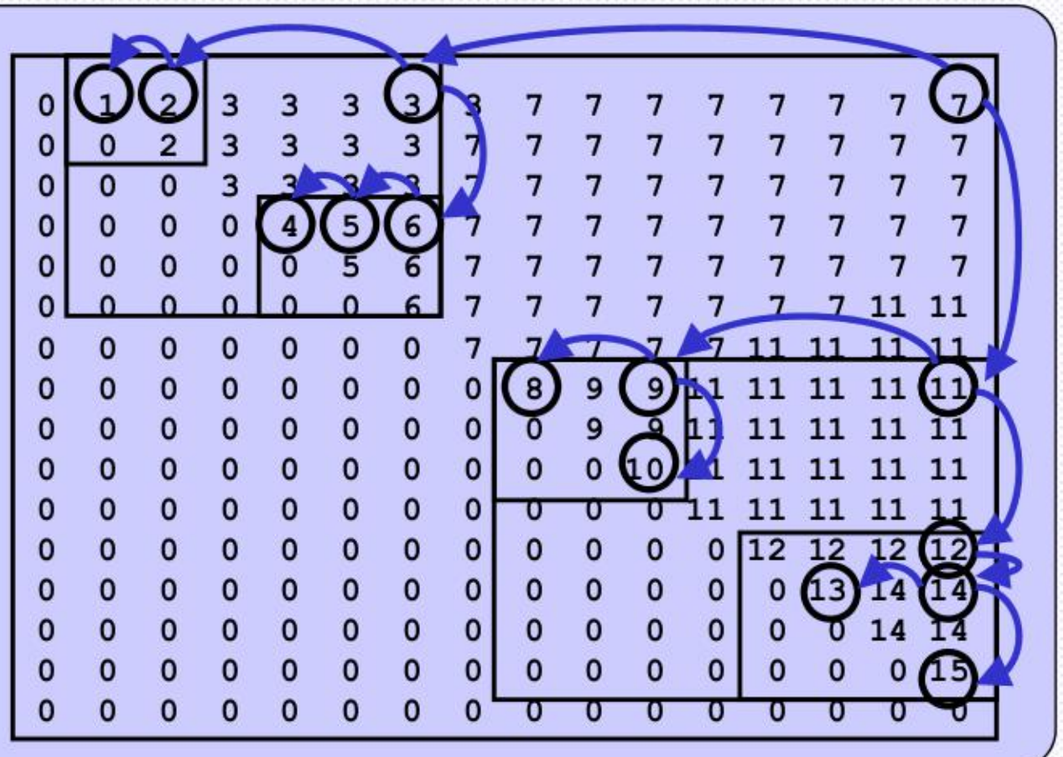
Kořeny optimálních podstromů

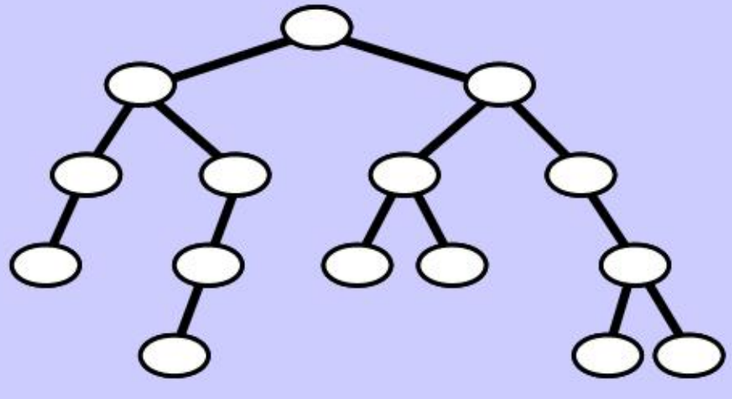
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	3	3	3	3	7	7	7	7	7	7	7	7
0	0	2	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	3	3	3	3	7	7	7	7	7	7	7	7	7
0	0	0	0	4	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	5	6	7	7	7	7	7	7	7	7	7
0	0	0	0	0	0	6	7	7	7	7	7	7	7	11	11
0	0	0	0	0	0	0	7	7	7	7	7	11	11	11	11
0	0	0	0	0	0	0	0	8	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	9	9	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	10	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	11	11	11	11	11
0	0	0	0	0	0	0	0	0	0	0	0	12	12	12	12
0	0	0	0	0	0	0	0	0	0	0	0	0	13	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

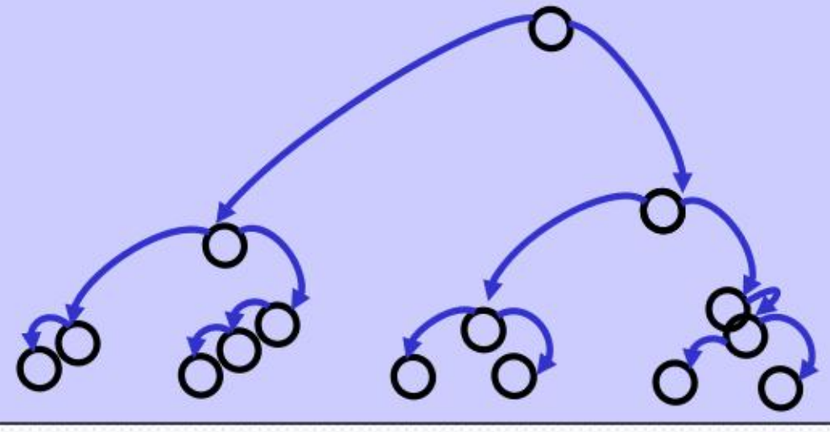
Výpočet optimálního BVS

Korespondence stromů



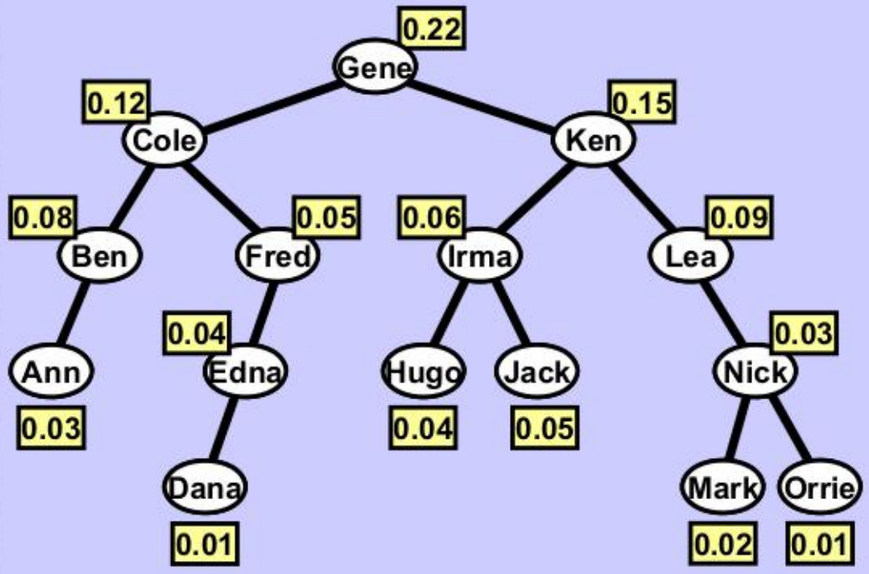


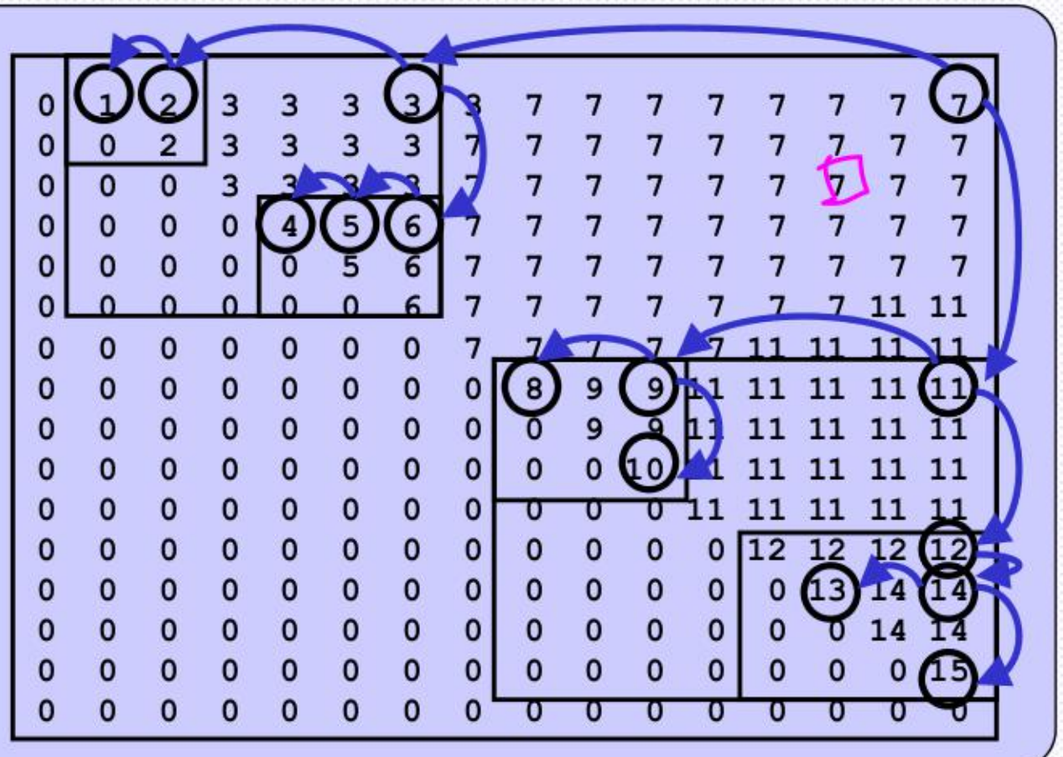


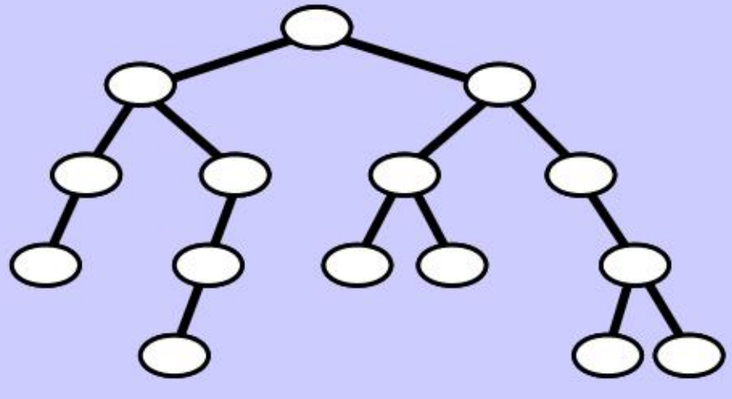


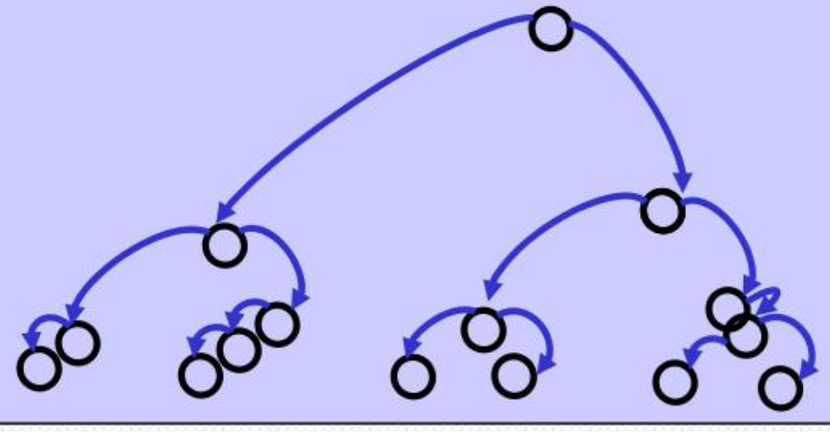
Výpočet optimálního BVS

Korespondence stromů









Výpočet optimálního BVS

Ceny optimálních podstromů

	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

Výpočet optimálního BVS

Ceny optimálních podstromů

	1-A	2-B	3-C	4-D	5-E	6-F	7-G	8-H	9-I	10-J	11-K	12-L	13-M	14-N	15-O
1-A	0.03	0.14	0.37	0.39	0.48	0.63	1.17	1.26	1.42	1.57	2.02	2.29	2.37	2.51	2.56
2-B	0	0.08	0.28	0.30	0.39	0.54	1.06	1.14	1.30	1.45	1.90	2.17	2.25	2.39	2.44
3-C	0	0	0.12	0.14	0.23	0.38	0.82	0.90	1.06	1.21	1.66	1.93	2.01	2.15	2.20
4-D	0	0	0	0.01	0.06	0.16	0.48	0.56	0.72	0.87	1.32	1.59	1.67	1.81	1.86
5-E	0	0	0	0	0.04	0.13	0.44	0.52	0.68	0.83	1.28	1.55	1.63	1.77	1.82
6-F	0	0	0	0	0	0.05	0.32	0.40	0.56	0.71	1.16	1.43	1.51	1.63	1.67
7-G	0	0	0	0	0	0	0.22	0.30	0.46	0.61	1.06	1.31	1.37	1.48	1.52
8-H	0	0	0	0	0	0	0	0.04	0.14	0.24	0.54	0.72	0.78	0.89	0.93
9-I	0	0	0	0	0	0	0	0	0.06	0.16	0.42	0.60	0.66	0.77	0.81
10-J	0	0	0	0	0	0	0	0	0	0.05	0.25	0.43	0.49	0.60	0.64
11-K	0	0	0	0	0	0	0	0	0	0	0.15	0.33	0.39	0.50	0.54
12-L	0	0	0	0	0	0	0	0	0	0	0	0.09	0.13	0.21	0.24
13-M	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.07	0.09
14-N	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0.05
15-O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01

Dynamické programování

Nejdelší společná podposloupnost

Nejdelší společná podposloupnost

Dvě posloupnosti

A: C B E A D D E A — |A| = 8

B: D E C D B D A — |B| = 7

Společná podposloupnost

A: C B E A D D E A

B: D E C D B D A

C: C D A — |C| = 3

Nejdelší společná podposloupnost (NSP)

A: C B E A D D E A

B: D E C D B D A

C: E D D A — |C| = 4

Nejdelší společná podposloupnost

Dvě posloupnosti

A: C B E A D D E A — |A| = 8

B: D E C D B D A — |B| = 7

Společná podposloupnost

A: C B E A D D E A

B: D E C D B D A

C: C D A — |C| = 3

Nejdelší společná podposloupnost (NSP)

A: C B E A D D E A

B: D E C D B D A

C: E D D A — |C| = 4

Nejdelší společná podposloupnost

A_n: $(a_1, a_2, ..., a_n)$

B_m: $(b_1, b_2, ..., b_m)$

C_k: $(c_1, c_2, ..., c_k)$

.....

$C_k = LCS(A_n, B_m)$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

Rekurzivní pravidla:

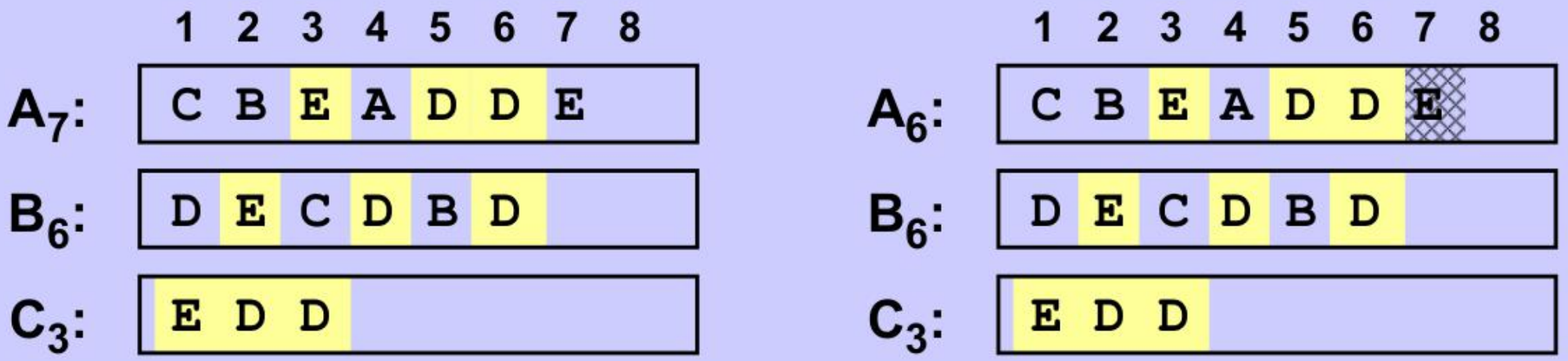
$(a_n = b_m) \Longrightarrow (c_k = a_n = b_m)$ & $(C_{k-1} = LCS (A_{n-1}, B_{m-1}))$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

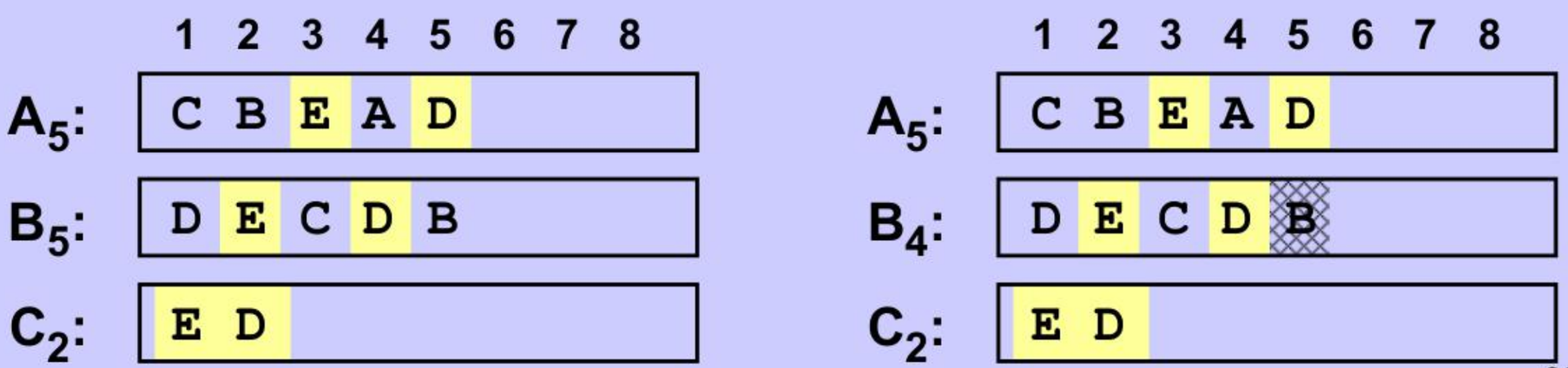
	1	2	3	4	5	6	7	8
A_7:	C	B	E	A	D	D	E	A
B_6:	D	E	C	D	B	D	A	
C_3:	E	D	D	A				

Nejdelší společná podposloupnost

$$(a_n != b_m) \ \& \ (c_k != a_n) \ ==> \ (C_k = LCS\ (A_{n-1}, B_m))$$



$$(a_n != b_m) \ \& \ (c_k != b_m) \ ==> \ (C_k = LCS\ (A_n, B_{m-1}))$$

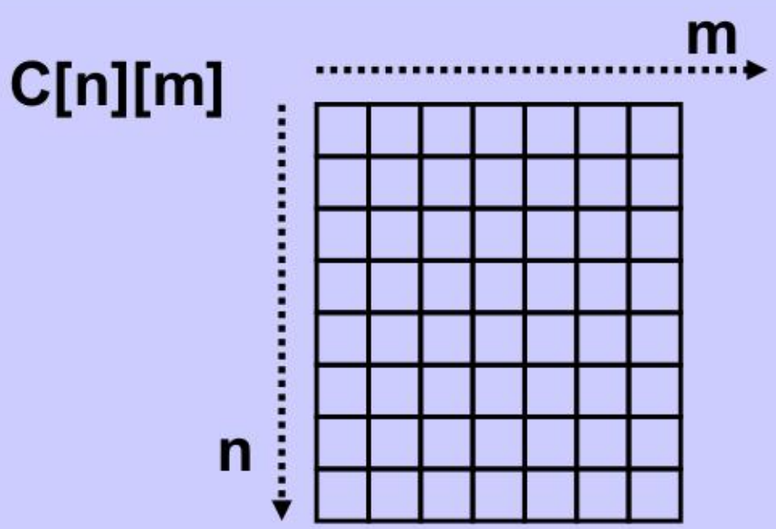


Nejdelší společná podposloupnost

Rekurzivní funkce – délka LCS

$$C(n,m) = \begin{cases} 0 & n = 0 \text{ or } m = 0 \\ C(n-1, m-1) + 1 & n > 0, m > 0, a_n = b_m \\ \max\{ C(n-1, m), C(n, m-1) \} & n > 0, m > 0, a_n \neq b_m \end{cases}$$

Strategie dynamického programování




```
for( a=1; a<=n; a++ )
  for( b=1; b<=m; b++ )
    C[a][b] = .... ;
}
```

Nejdelší společná podposloupnost

Konstrukce DP tabulek pro LCS

```
void findLCS() {
  for( int a=1; a<=n; a++ )
    for( int b=1; b<=m; b++ )
      if( A[a] == B[b] ) {
          C[a][b] = C[a-1][b-1]+1;
          arrows[a][b] = DIAG; ↖
          }
      else
        if( C[a-1][b] > C[a][b-1] ) {
           C[a][b] = C[a-1][b];
           arrows[a][b] = UP; ↑
        }
        else {
          C[a][b] = C[a][b-1];
          arrows[a][b] = LEFT; ←
}       }
```

Nejdelší společná podposloupnost

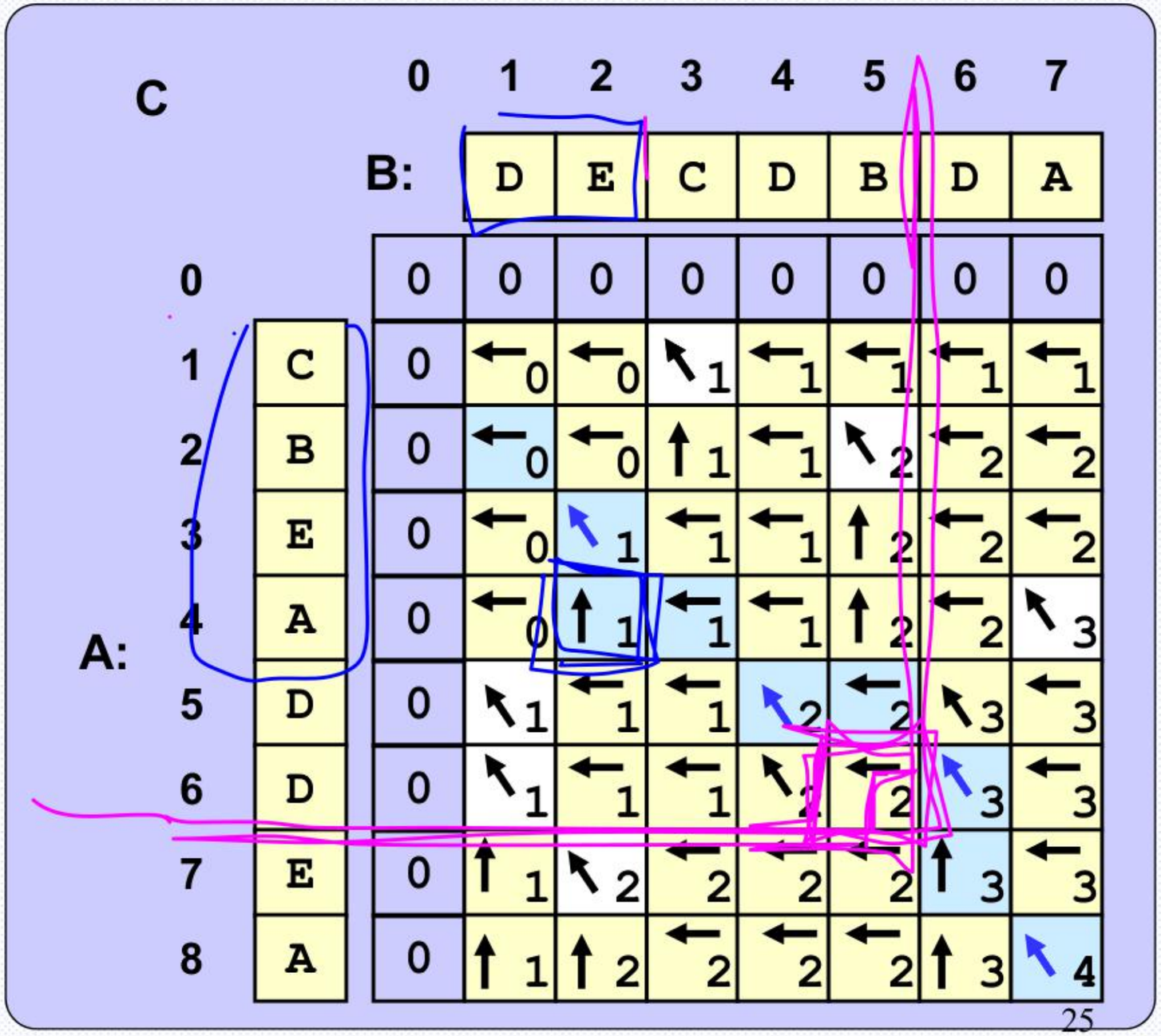
Pole NSP pro “CBEADDEA” a “DECDBDA”

			C	0	1	2	3	4	5	6	7
			B:	D	E	C	D	B	D	A
	0		0	0	0	0	0	0	0	0
	1	C	0	←0	←0	↖1	←1	←1	←1	←1
	2	B	0	←0	←0	↑1	←1	↖2	←2	←2
	3	E	0	←0	↖1	←1	←1	↑2	←2	←2
A:	4	A	0	←0	↑1	←1	←1	↑2	←2	↖3
	5	D	0	↖1	←1	←1	↖2	←2	↖3	←3
	6	D	0	↖1	←1	←1	↖2	←2	↖3	←3
	7	E	0	↑1	↖2	←2	←2	←2	↑3	←3
	8	A	0	↑1	↑2	←2	←2	←2	↑3	↖4

Nejdelší společná podposloupnost

Pole NSP pro

“CBEADDEA”
a
“DECDBDA”



Nejdelší společná podposloupnost

Konstrukce DP tabulek pro LCS

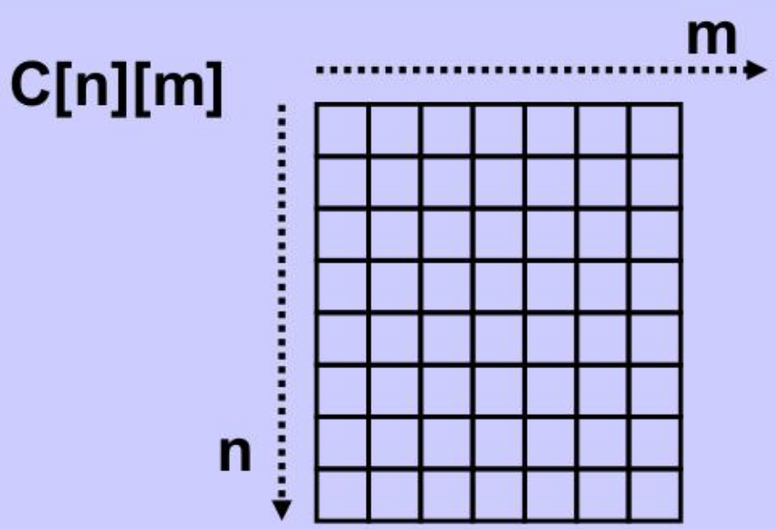
```
void findLCS() {
  for( int a=1; a<=n; a++ )
    for( int b=1; b<=m; b++ )
      if( A[a] == B[b] ) {
          C[a][b] = C[a-1][b-1]+1;
          arrows[a][b] = DIAG; ↖
          }
      else
        if( C[a-1][b] > C[a][b-1] ) {
           C[a][b] = C[a-1][b];
           arrows[a][b] = UP; ↑
        }
        else {
          C[a][b] = C[a][b-1];
          arrows[a][b] = LEFT; ←
}       }
```

Nejdelší společná podposloupnost

Rekurzivní funkce – délka LCS

$$C(n,m) = \begin{cases} 0 & n = 0 \text{ or } m = 0 \\ C(n-1, m-1) + 1 & n > 0, m > 0, a_n = b_m \\ \max\{ C(n-1, m), C(n, m-1) \} & n > 0, m > 0, a_n \neq b_m \end{cases}$$

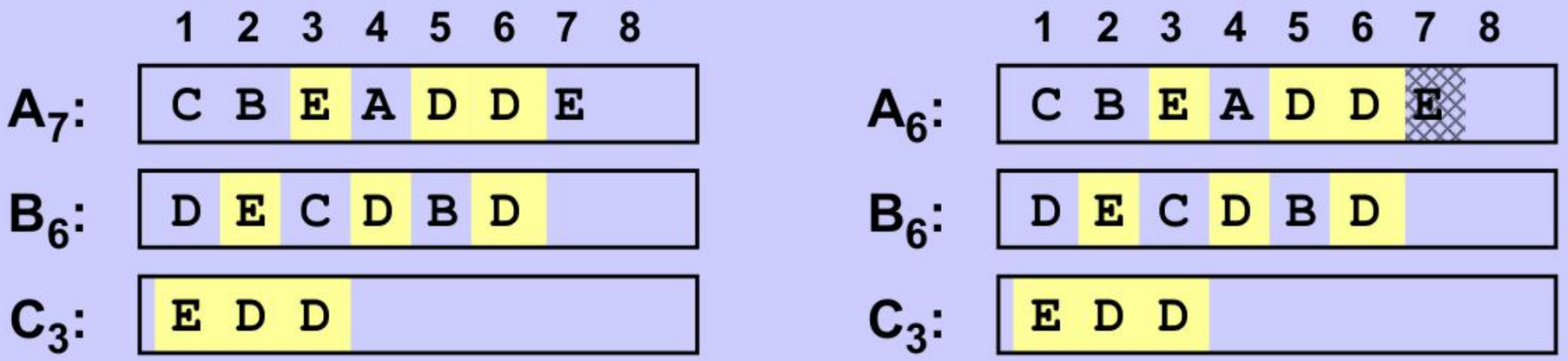
Strategie dynamického programování



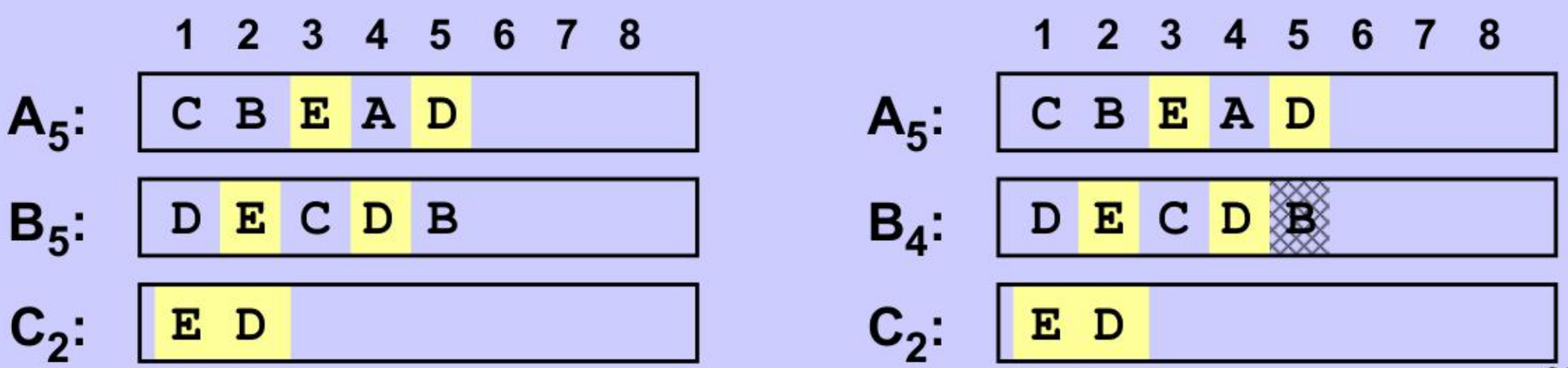

```
for( a=1; a<=n; a++ )
  for( b=1; b<=m; b++ )
    C[a][b] = .... ;
}
```

Nejdelší společná podposloupnost

$$(a_n != b_m) \ \& \ (c_k != a_n) \ ==> \ (C_k = LCS\ (A_{n-1}, B_m))$$



$$(a_n != b_m) \ \& \ (c_k != b_m) \ ==> \ (C_k = LCS\ (A_n, B_{m-1}))$$



Nejdelší společná podposloupnost

A_n: $(a_1, a_2, ..., a_n)$

B_m: $(b_1, b_2, ..., b_m)$

C_k: $(c_1, c_2, ..., c_k)$

.....

$C_k = LCS(A_n, B_m)$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

Rekurzivní pravidla:

$(a_n = b_m) \Longrightarrow (c_k = a_n = b_m)$ & $(C_{k-1} = LCS (A_{n-1}, B_{m-1}))$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

	1	2	3	4	5	6	7	8
A_7:	C	B	E	A	D	D	E	A
B_6:	D	E	C	D	B	D	A	
C_3:	E	D	D	A				

Nejdelší společná podposloupnost

Dvě posloupnosti

A: C B E A D D E A — |A| = 8

B: D E C D B D A — |B| = 7

Společná podposloupnost

A: C B E A D D E A

B: D E C D B D A

C: C D A — |C| = 3

Nejdelší společná podposloupnost (NSP)

A: C B E A D D E A

B: D E C D B D A

C: E D D A — |C| = 4

Nejdelší společná podposloupnost

Dvě posloupnosti

A: C B E A D D E A — |A| = 8

B: D E C D B D A — |B| = 7

Společná podposloupnost

A: C B E A D D E A

B: D E C D B D A

C: C D A — |C| = 3

Nejdelší společná podposloupnost (NSP)

A: C B E A D D E A

B: D E C D B D A

C: E D D A — |C| = 4

Nejdelší společná podposloupnost

A_n: $(a_1, a_2, ..., a_n)$

B_m: $(b_1, b_2, ..., b_m)$

C_k: $(c_1, c_2, ..., c_k)$

.....

$C_k = LCS(A_n, B_m)$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

Rekurzivní pravidla:

$(a_n = b_m) \Longrightarrow (c_k = a_n = b_m)$ & $(C_{k-1} = LCS (A_{n-1}, B_{m-1}))$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

	1	2	3	4	5	6	7	8
A_7:	C	B	E	A	D	D	E	A
B_6:	D	E	C	D	B	D	A	
C_3:	E	D	D	A				

Nejdelší společná podposloupnost

A_n: $(a_1, a_2, ..., a_n)$

B_m: $(b_1, b_2, ..., b_m)$

C_k: $(c_1, c_2, ..., c_k)$

.....

$C_k = LCS(A_n, B_m)$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

Rekurzivní pravidla:

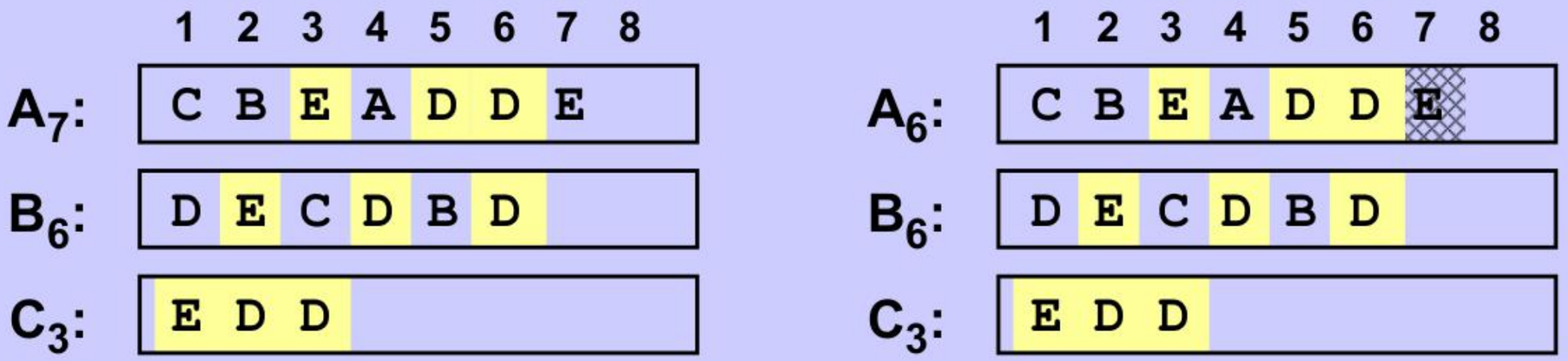
$(a_n = b_m) \Longrightarrow (c_k = a_n = b_m) \ \& \ (C_{k-1} = LCS (A_{n-1}, B_{m-1}))$

	1	2	3	4	5	6	7	8
A_8:	C	B	E	A	D	D	E	A
B_7:	D	E	C	D	B	D	A	
C_4:	E	D	D	A				

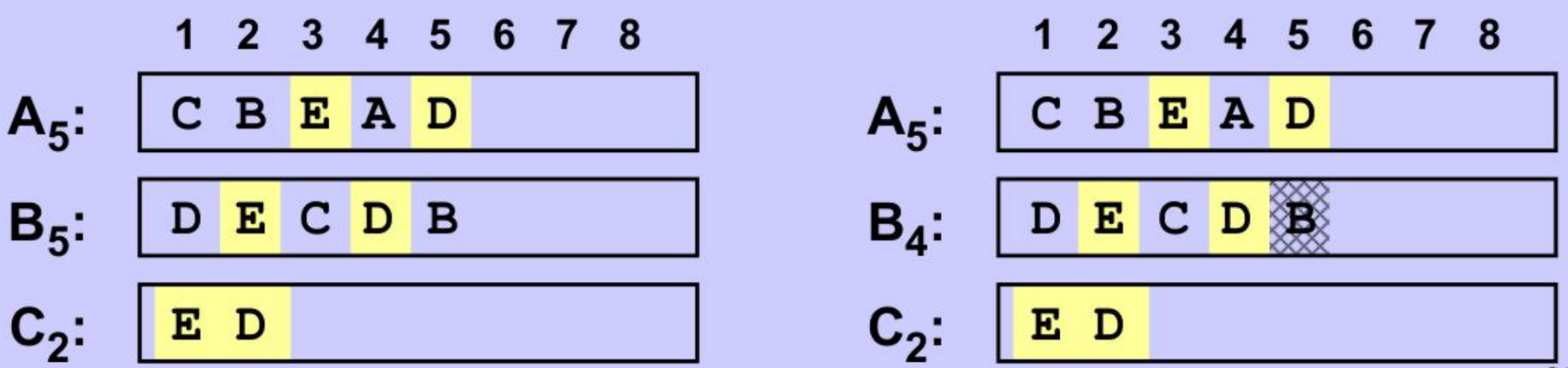
	1	2	3	4	5	6	7	8
A_7:	C	B	E	A	D	D	E	A
B_6:	D	E	C	D	B	D	A	
C_3:	E	D	D	A				

Nejdelší společná podposloupnost

$$(a_n != b_m) \ \& \ (c_k != a_n) \ ==> \ (C_k = LCS\ (A_{n-1}, B_m))$$

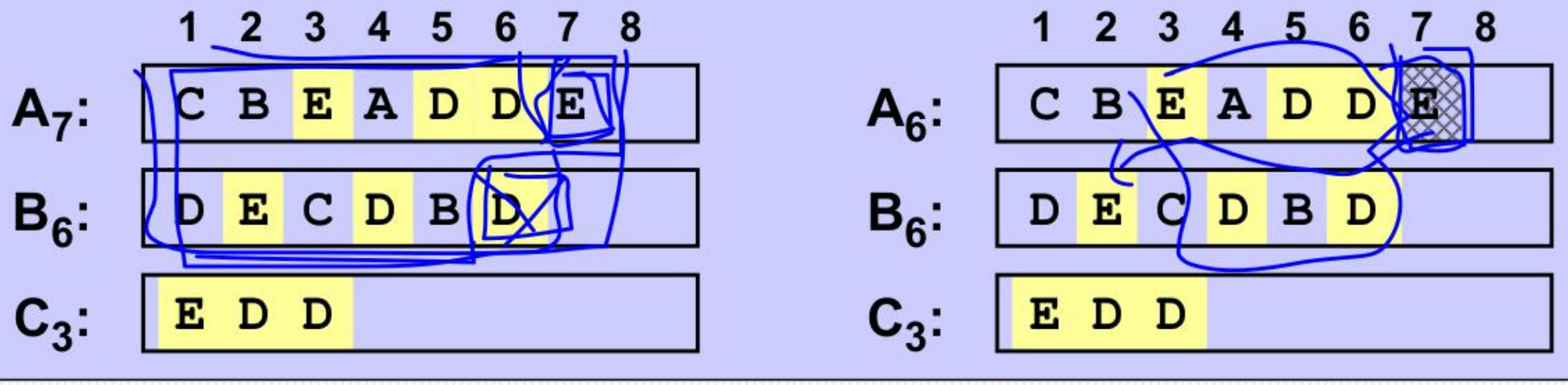


$$(a_n != b_m) \ \& \ (c_k != b_m) \ ==> \ (C_k = LCS\ (A_n, B_{m-1}))$$

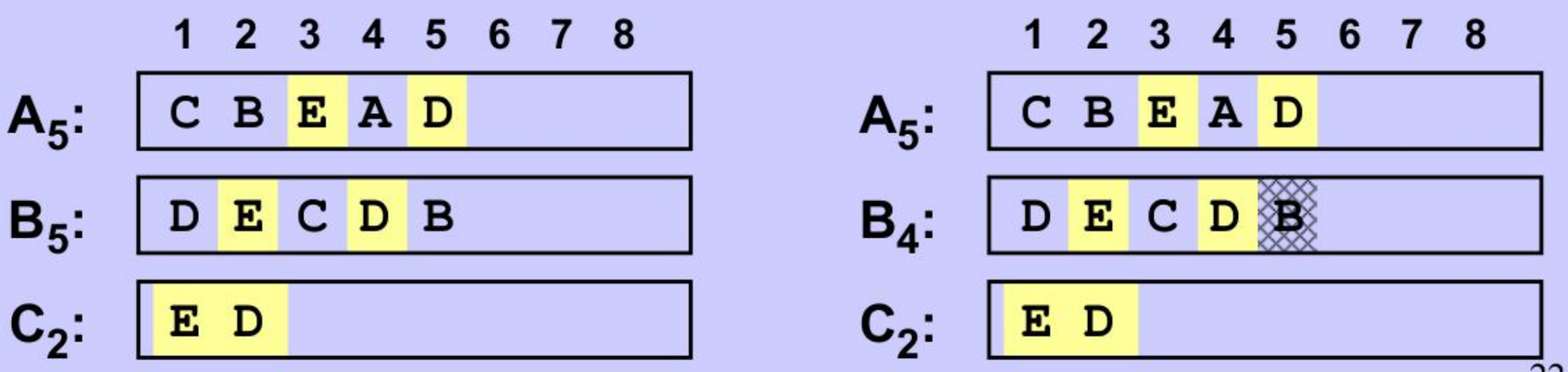


Nejdelší společná podposloupnost

$$(a_n != b_m) \ \& \ (c_k != a_n) \ ==> \ (C_k = LCS\ (A_{n-1}, B_m))$$



$$(a_n != b_m) \ \& \ (c_k != b_m) \ ==> \ (C_k = LCS\ (A_n, B_{m-1}))$$

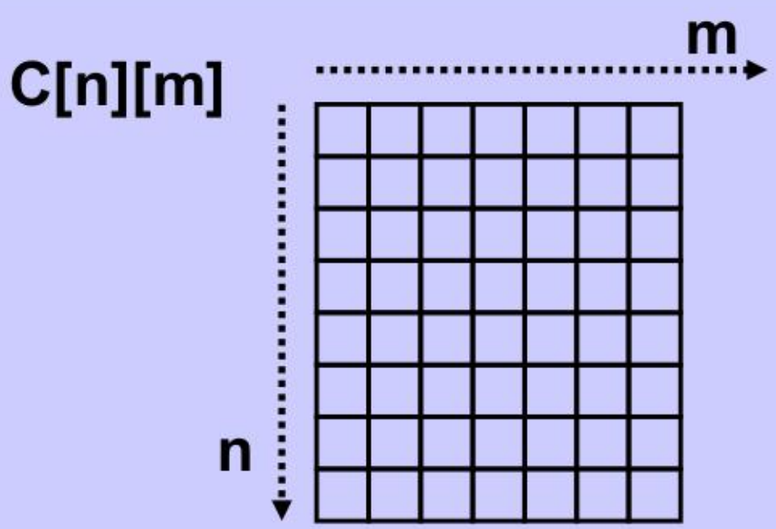


Nejdelší společná podposloupnost

Rekurzivní funkce – délka LCS

$$C(n,m) = \begin{cases} 0 & n = 0 \text{ or } m = 0 \\ C(n-1, m-1) + 1 & n > 0, m > 0, a_n = b_m \\ \max\{ C(n-1, m), C(n, m-1) \} & n > 0, m > 0, a_n \neq b_m \end{cases}$$

Strategie dynamického programování



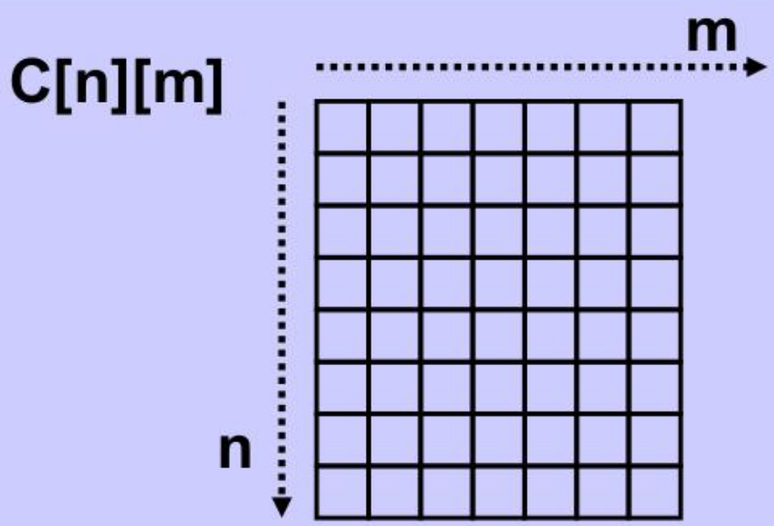

```
for( a=1; a<=n; a++ )
  for( b=1; b<=m; b++ )
    C[a][b] = .... ;
}
```

Nejdelší společná podposloupnost

Rekurzivní funkce – délka LCS

$$C(n,m) = \begin{cases} 0 & n = 0 \text{ or } m = 0 \\ C(n-1, m-1) + 1 & n > 0, m > 0, a_n = b_m \\ \max\{ C(n-1, m), C(n, m-1) \} & n > 0, m > 0, a_n \neq b_m \end{cases}$$

Strategie dynamického programování




```
for( a=1; a<=n; a++ )
  for( b=1; b<=m; b++ )
    C[a][b] = .... ;
}
```

Nejdelší společná podposloupnost

Konstrukce DP tabulek pro LCS

```
void findLCS() {
  for( int a=1; a<=n; a++ )
    for( int b=1; b<=m; b++ )
      if( A[a] == B[b] ) {
          C[a][b] = C[a-1][b-1]+1;
          arrows[a][b] = DIAG; ↖
          }
      else
        if( C[a-1][b] > C[a][b-1] ) {
           C[a][b] = C[a-1][b];
           arrows[a][b] = UP; ↑
        }
        else {
          C[a][b] = C[a][b-1];
          arrows[a][b] = LEFT; ←
}       }
```

Nejdelší společná podposloupnost

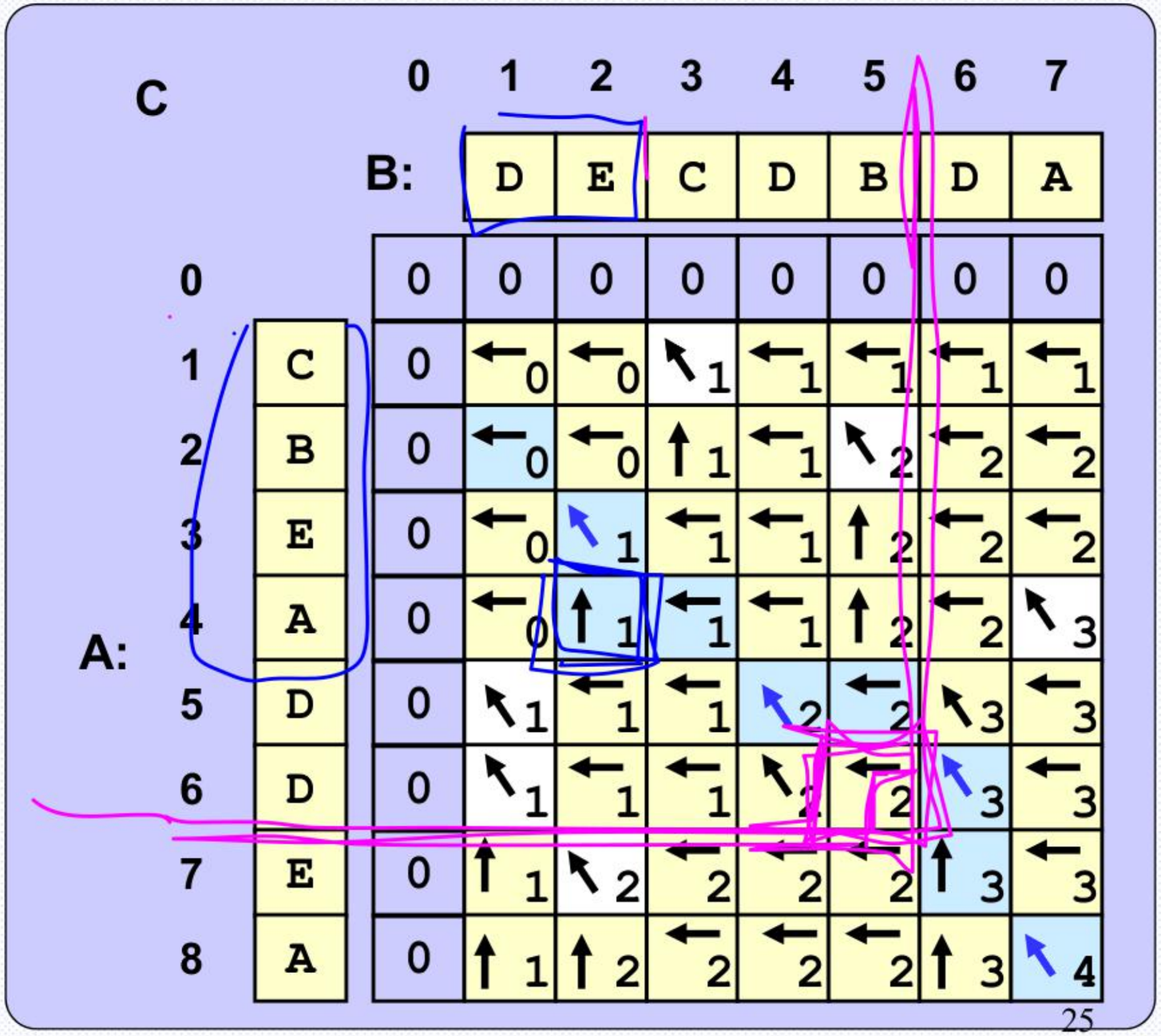
Konstrukce DP tabulek pro LCS

```
void findLCS() {
  for( int a=1; a<=n; a++ )
    for( int b=1; b<=m; b++ )
      if( A[a] == B[b] ) {
          C[a][b] = C[a-1][b-1]+1;
          arrows[a][b] = DIAG;
          }
      else
        if( C[a-1][b] > C[a][b-1] ) {
            C[a][b] = C[a-1][b];
            arrows[a][b] = UP;
        }
        else {
          C[a][b] = C[a][b-1];
          arrows[a][b] = LEFT;
}       }
```

Nejdelší společná podposloupnost

Pole NSP pro

“CBEADDEA”
a
“DECDBDA”



Nejdelší společná podposloupnost

Pole NSP pro

“CBEADDEA”
a
“DECDBDA”

			B:	D	E	C	D	B	D	A
			0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0	0
	C	1	0	←0	←0	↖1	←1	←1	←1	←1
	B	2	0	←0	←0	↑1	←1	↖2	←2	←2
	E	3	0	←0	↖1	←1	←1	↑2	←2	←2
A:	A	4	0	←0	↑1	←1	←1	↑2	←2	↖3
	D	5	0	↖1	←1	←1	↖2	←2	↖3	←3
	D	6	0	↖1	←1	←1	↖2	←2	↖3	←3
	E	7	0	↑1	↖2	←2	←2	←2	↑3	←3
	A	8	0	↑1	↑2	←2	←2	←2	↑3	↖4

Nejdelší společná podposloupnost

Výpis NSP -- rekurzivně :)

```
void outLCS( int a, int b ) {
if( a ==0 || b == 0 ) return;

if( arrows[a][b]  == DIAG ) {
    outLCS(a-1, b-1);        // recursion ...
    print(A[a]);             // ... reverses the sequence!
    }
else
    if( arrows[a][b] == UP )
        outLCS(a-1,b);
    else
        outLCS(a,b-1);
}
```

ALG 11

Dynamické programování

- Úloha batohu neomezená
- Úloha batohu 0/1

Úloha batohu / Knapsack problem

Máme N předmětů, každý s váhou Vi a cenou Ci (i = 1, 2, ..., N)
a batoh s kapacitou váhy K.
Máme naložit batoh těmito předměty tak,
aby kapacita K nebyla překročena a obsah měl maximální cenu.

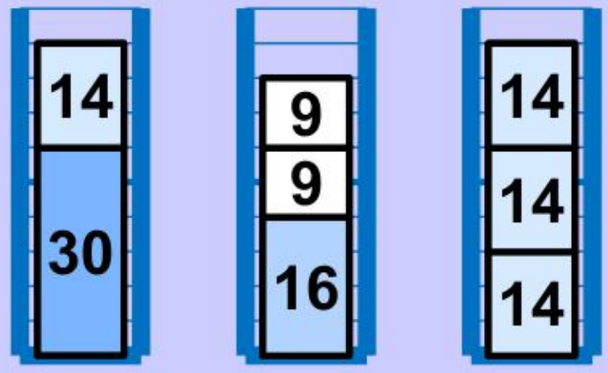
Neomezená varianta -- Každý předmět lze použít libovolněkrát.

0/1 varianta -- Každý předmět lze použít nejvýše jednou.

Schematický batoh s kapacitou 10

Předměty s uvedenou cenou, váha ~ výška

Několik možných konfigurací



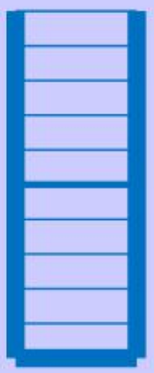
Úloha batohu / Knapsack problem

Máme N předmětů, každý s váhou Vi a cenou Ci (i = 1, 2, ..., N)
a batoh s kapacitou váhy K.
Máme naložit batoh těmito předměty tak,
aby kapacita K nebyla překročena a obsah měl maximální cenu.

Neomezená varianta -- Každý předmět lze použít libovolněkrát.

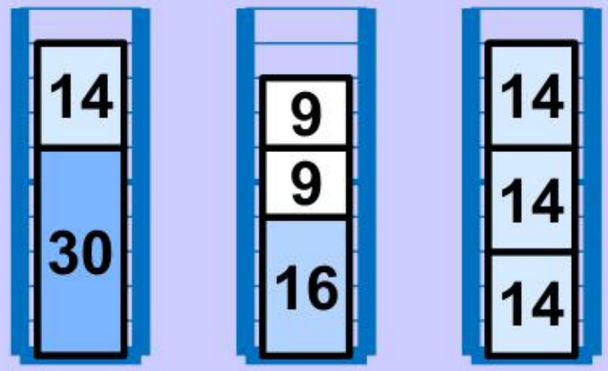
0/1 varianta -- Každý předmět lze použít nejvýše jednou.

Schematický batoh s kapacitou 10



Předměty s uvedenou cenou, váha ~ výška

Několik možných konfigurací

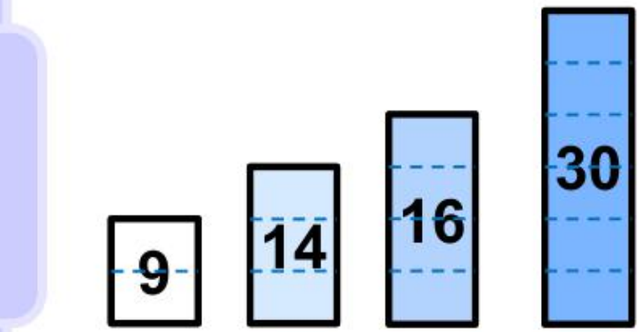


Neomezená úloha batohu

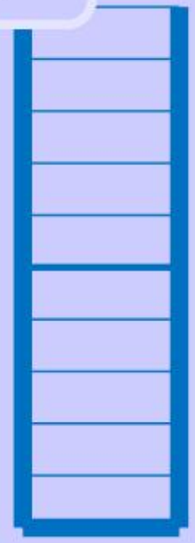
Příklad

N = 4

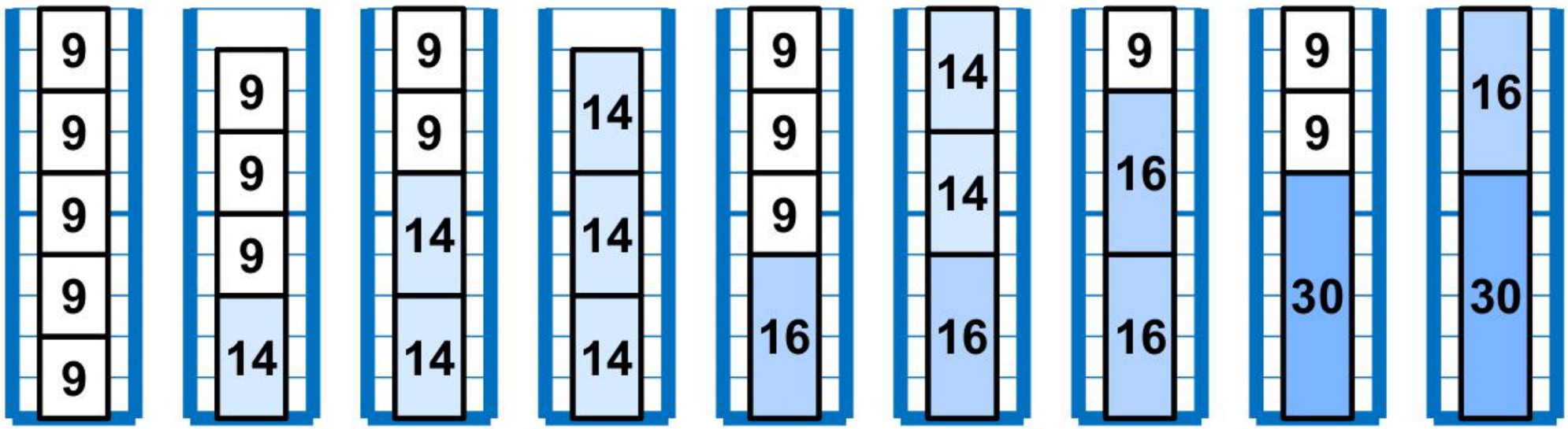
Váha	2	3	4	6
Cena	9	14	16	30



Batoh s kapacitou 10



Některé možnosti naplnění a odpovídající ceny



Neomezená úloha batohu

Použijeme K+1 batohů, o kapacitách 0, 1, 2, 3, ..., K.
Hodnotu optimálního naplnění batohu s kapacitou K
lze získat jako maximum z hodnot

-- (optimalní naplnění batohu o kapacitě K - V1) + C1,
-- (optimalní naplnění batohu o kapacitě K - V2) + C2,
...
-- (optimalní naplnění batohu o kapacitě K - VN) + CN.

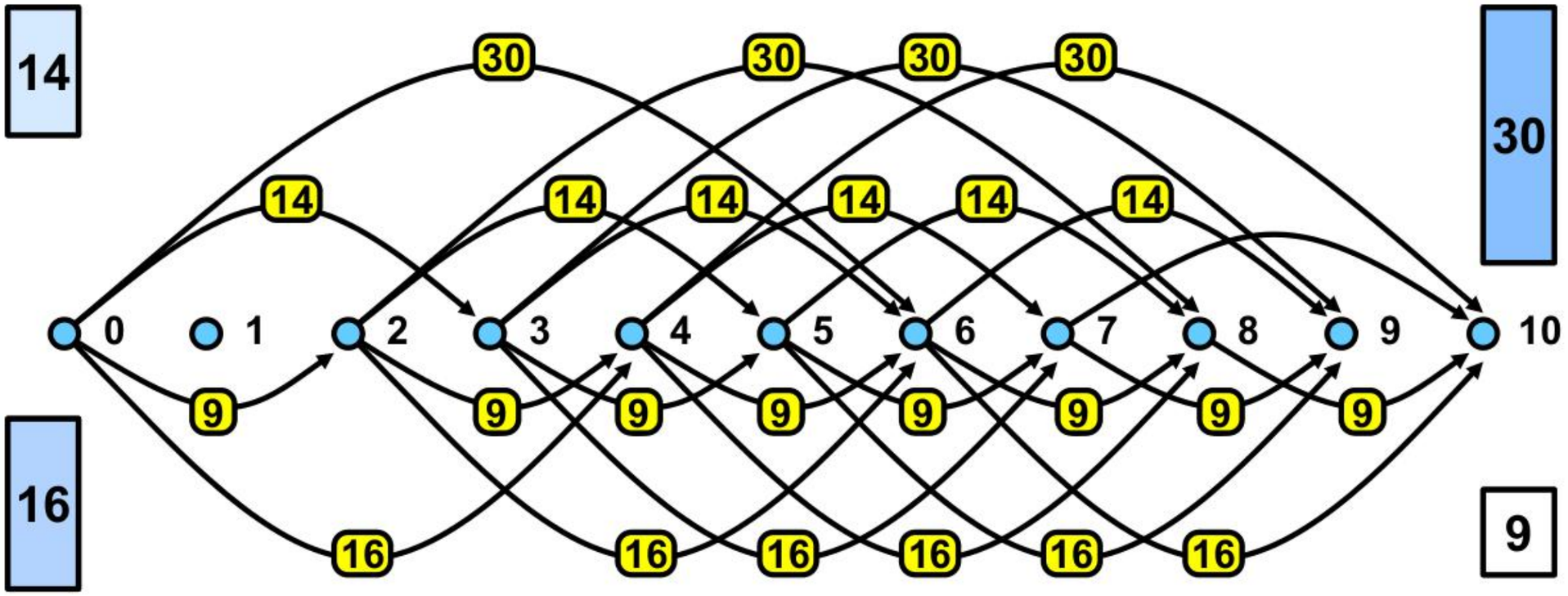
Optimální naplnění batohu o kapacitě K - Vi (i = 1..N) je stejnou úlohou, jen s menšími daty. Hodnoty předpočítáme standardně metodou DP do 1D tabulky.

Neomezenou úlohu batohu lze přímo vyjádřit jako úlohu nalezení nejdelší cesty v DAG. Postup řešení je identický.

Neomezená úloha batohu -- převod na DAG

DAG:
Uzly: Kapacity 0, 1, 2, 3, ..., K.
Hrany: Z uzlu X vedou hrany po řadě do uzlů X+V1, X+V2, ..., V+VN, jsou po řadě ohodnoceny cenami C1, C2, ..., CN.

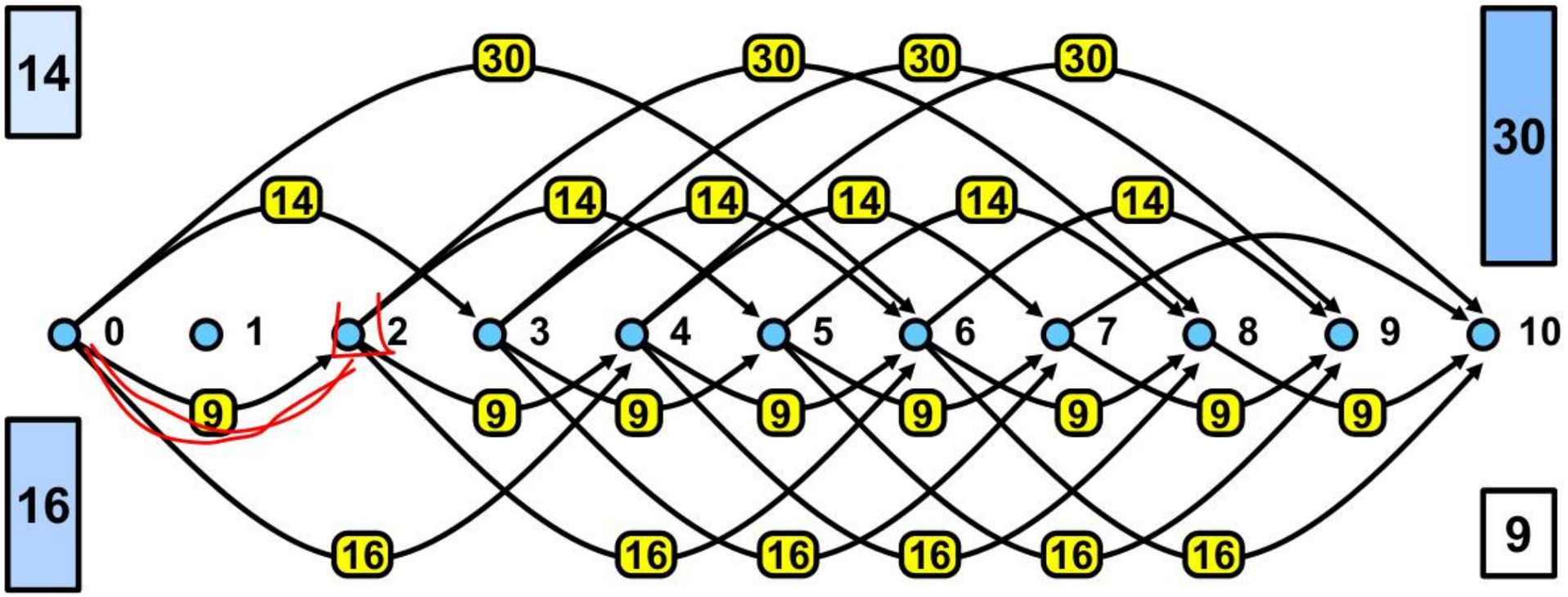
Příklad K = 10, N = 4, Vi = (2, 3, 4, 6), Ci = (9, 14, 16, 30), i = 1..4.



Neomezená úloha batohu -- převod na DAG

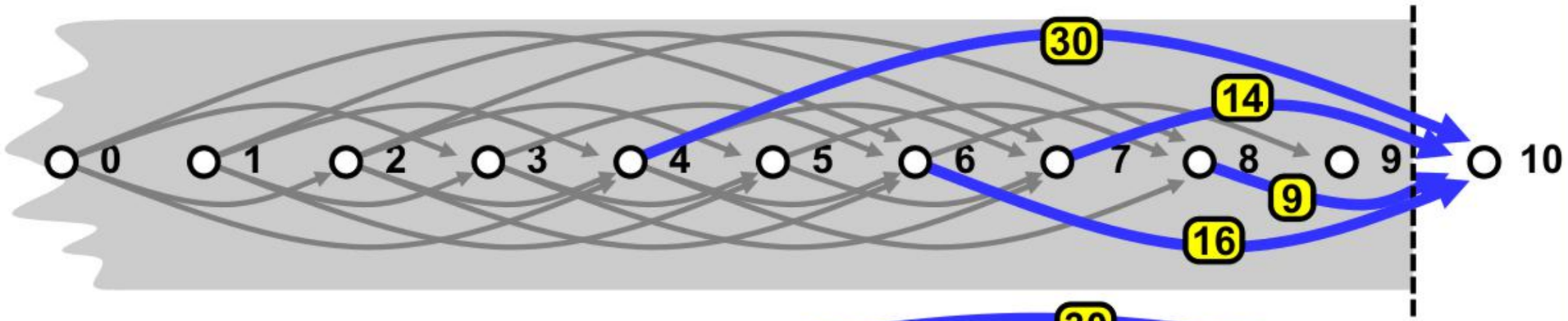
DAG:
Uzly: Kapacity 0, 1, 2, 3, ..., K.
Hrany: Z uzlu X vedou hrany po řadě do uzlů X+V1, X+V2, ..., V+VN, jsou po řadě ohodnoceny cenami C1, C2, ..., CN.

Příklad K = 10, N = 4, Vi = (2, 3, 4, 6), Ci = (9, 14, 16, 30), i = 1..4.

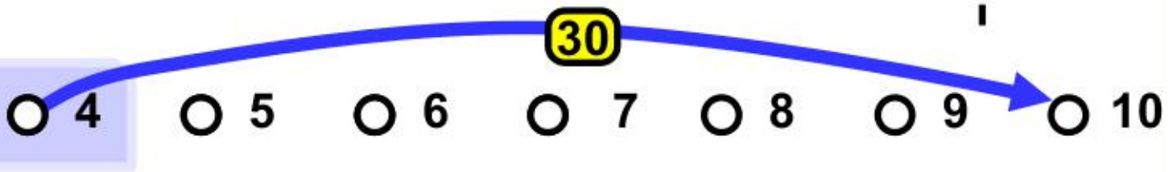


Neomezená úloha batohu -- jako DAG

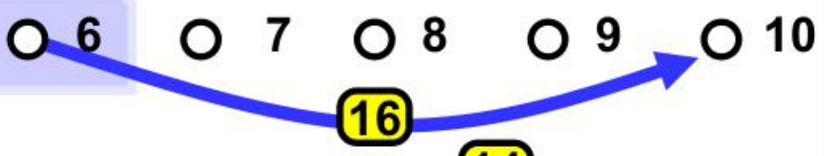
Optimální naplnění kapacity 10 = ??



Opt. naplnění kapacity 4 = 18

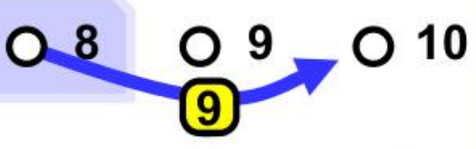


Optimální naplnění kapacity 6 = 30



Optimální naplnění kapacity 7 = 32

Optimální naplnění kapacity 8 = 39

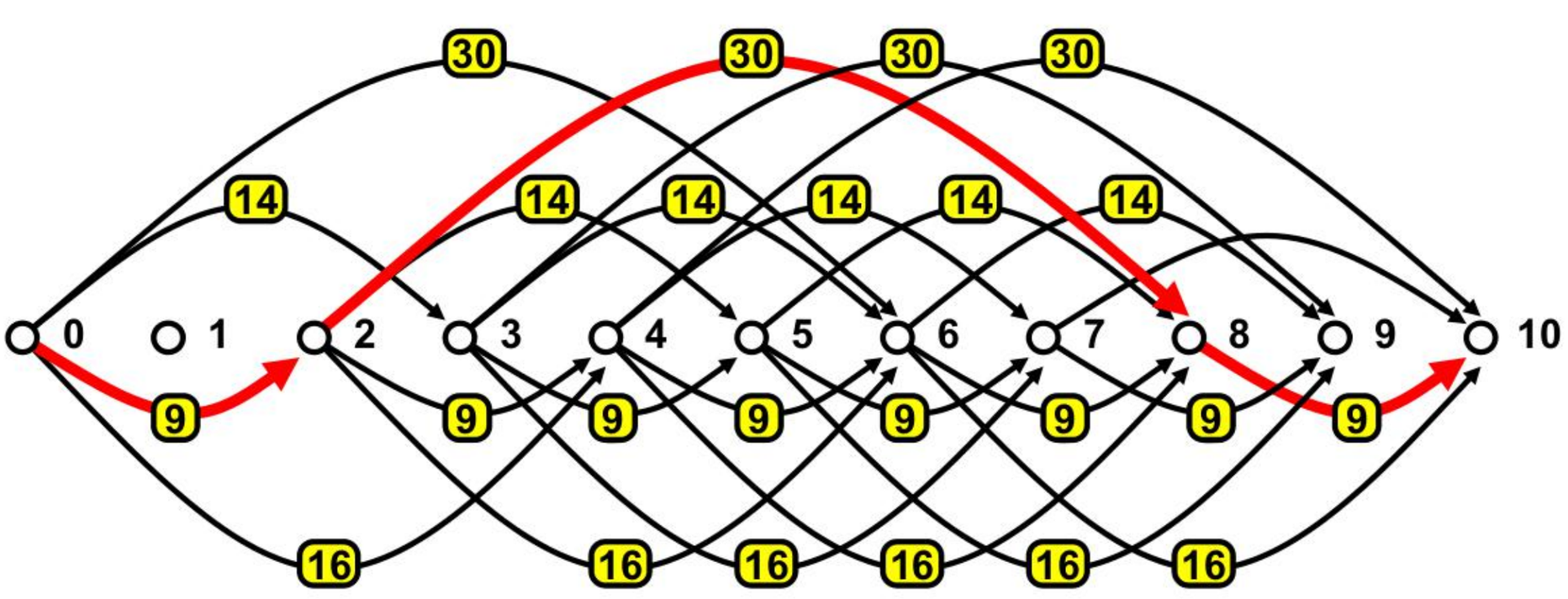


Optimální naplnění kapacity 10 = max(18 + 30, 30 + 16, 32 + 14, 39 + 9) = 48

Neomezená úloha batohu

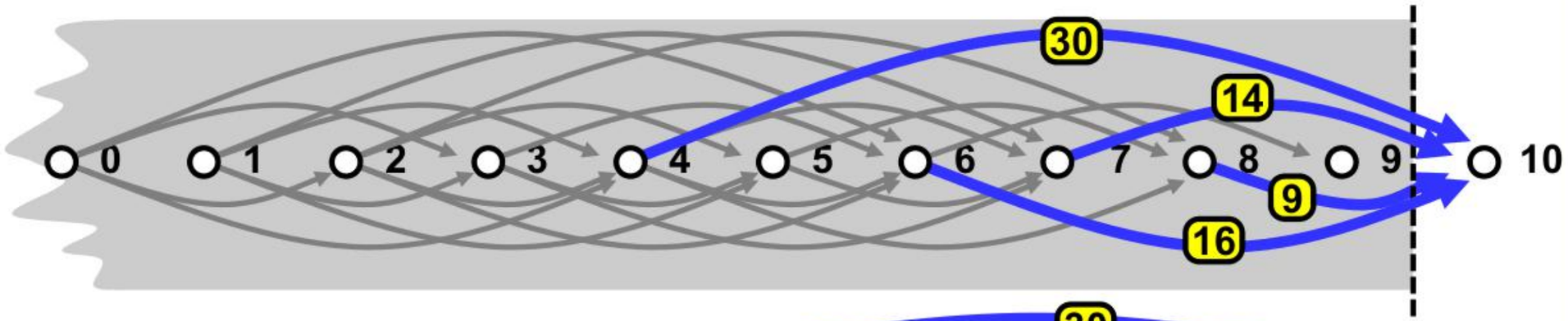
Nejdelší cesta odpovídá optimálnímu naplnění batohu.
Dvě hrany s cenou 9 a jedna hrana s cenou 30, celkem cena = 48.

Batoh optimálně naplníme dvěma předměty s váhou 2 a cenou 9 a jedním předmětem s váhou 6 a cenou 30.

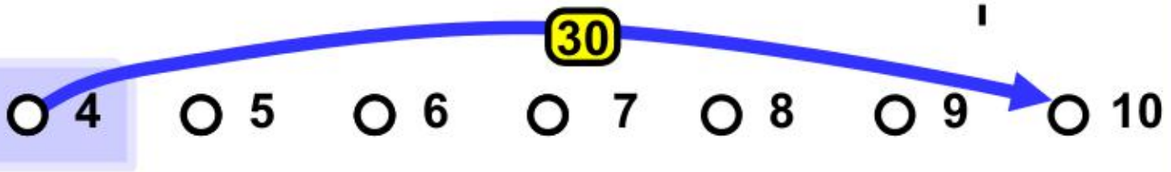


Neomezená úloha batohu -- jako DAG

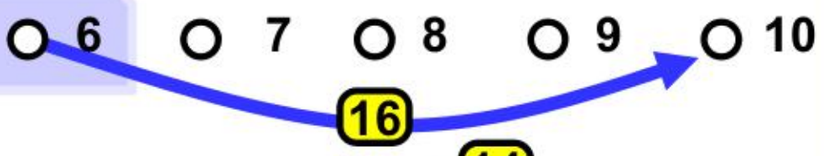
Optimální naplnění kapacity 10 = ??



Opt. naplnění kapacity 4 = 18

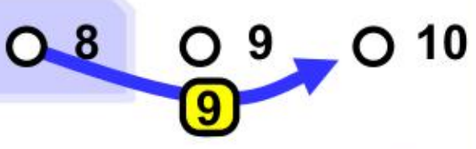


Optimální naplnění kapacity 6 = 30



Optimální naplnění kapacity 7 = 32

Optimální naplnění kapacity 8 = 39

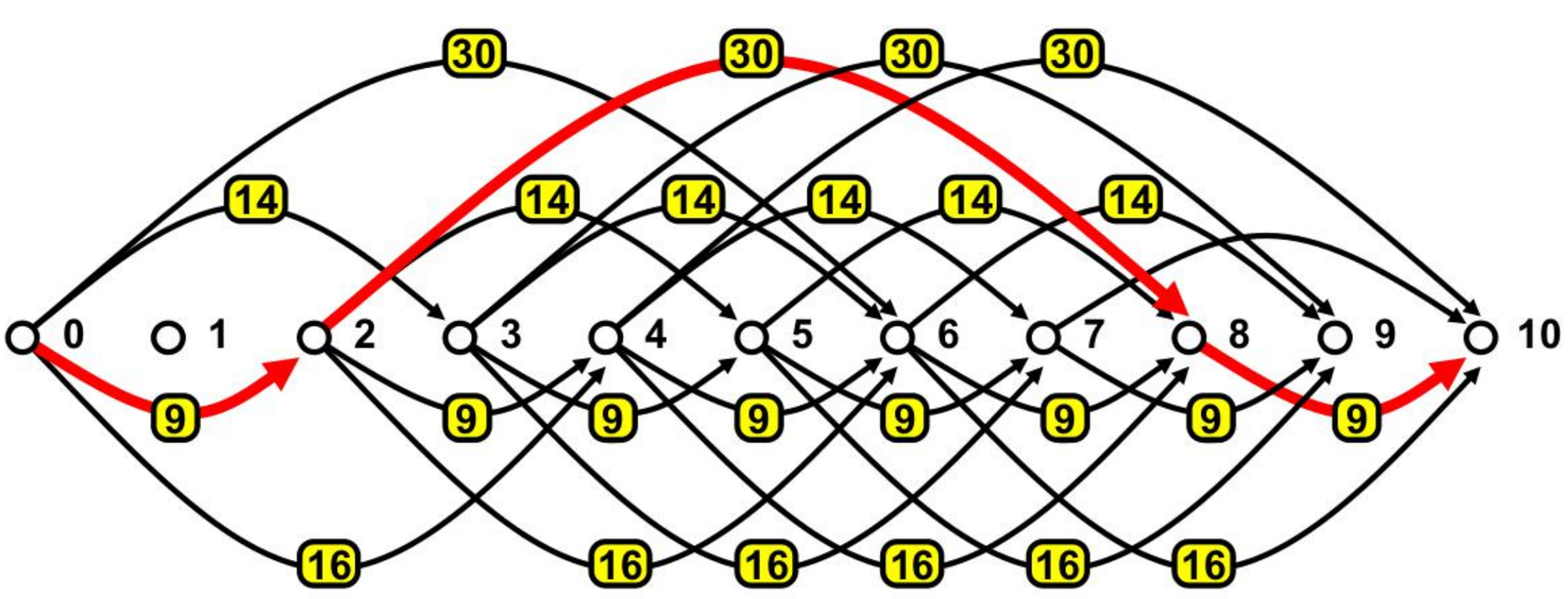


Optimální naplnění kapacity 10 = max(18 + 30, 30 + 16, 32 + 14, 39 + 9) = 48

Neomezená úloha batohu

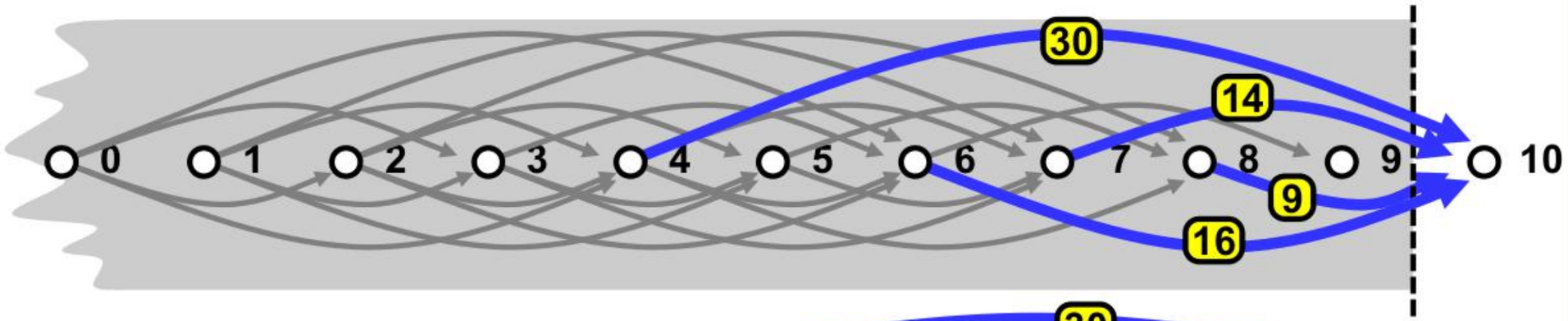
Nejdelší cesta odpovídá optimálnímu naplnění batohu.
Dvě hrany s cenou 9 a jedna hrana s cenou 30, celkem cena = 48.

Batoh optimálně naplníme dvěma předměty s váhou 2 a cenou 9 a jedním předmětem s váhou 6 a cenou 30.

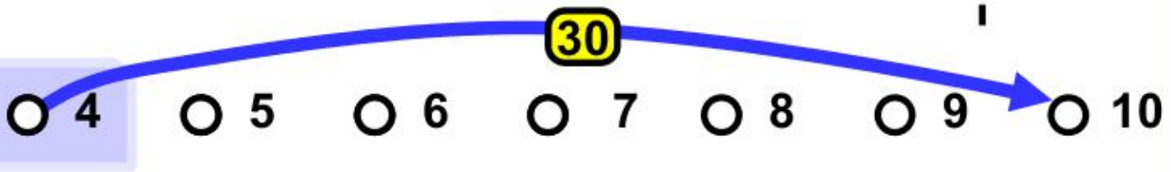


Neomezená úloha batohu -- jako DAG

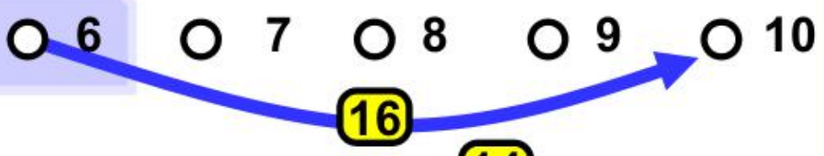
Optimální naplnění kapacity 10 = ??



Opt. naplnění kapacity 4 = 18

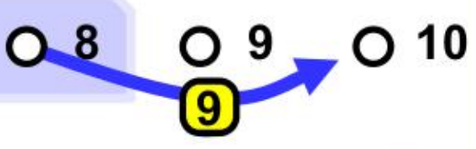


Optimální naplnění kapacity 6 = 30



Optimální naplnění kapacity 7 = 32

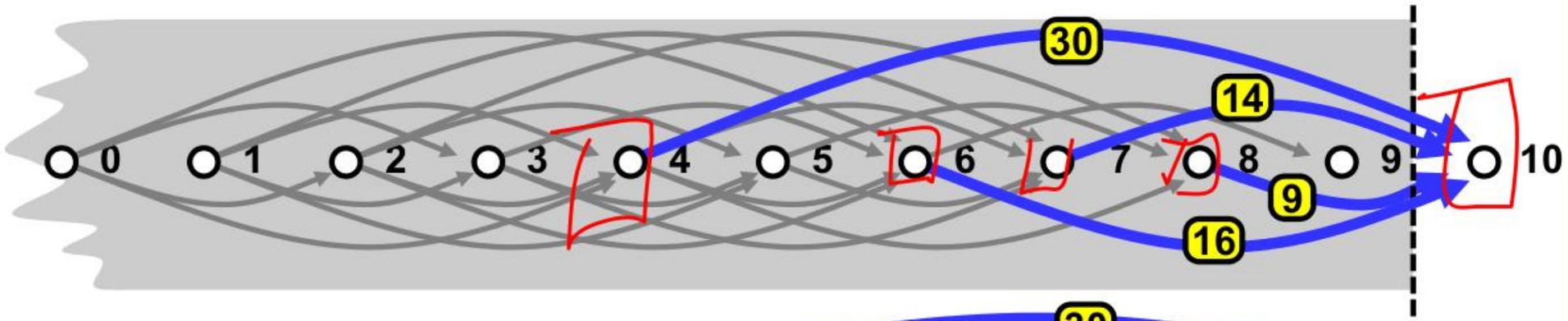
Optimální naplnění kapacity 8 = 39



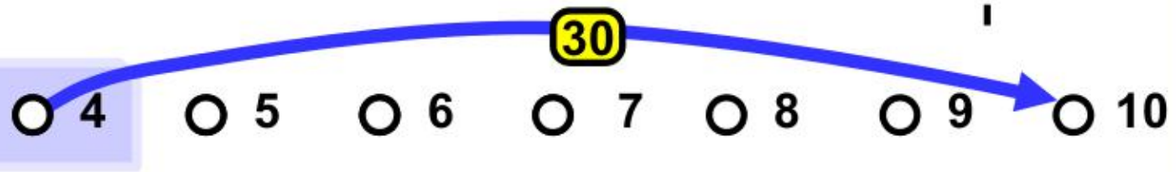
Optimální naplnění kapacity 10 = max(18 + 30, 30 + 16, 32 + 14, 39 + 9) = 48

Neomezená úloha batohu -- jako DAG

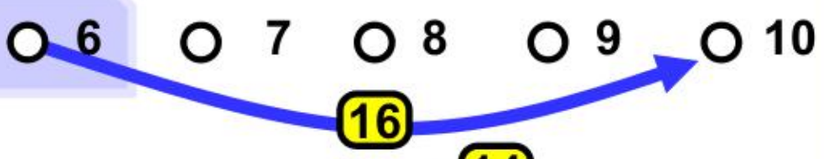
Optimální naplnění kapacity 10 = ??



Opt. naplnění kapacity 4 = 18

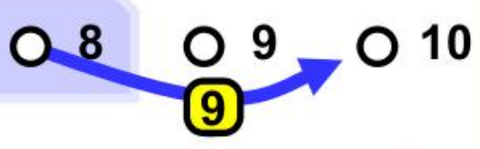


Optimální naplnění kapacity 6 = 30



Optimální naplnění kapacity 7 = 32

Optimální naplnění kapacity 8 = 39

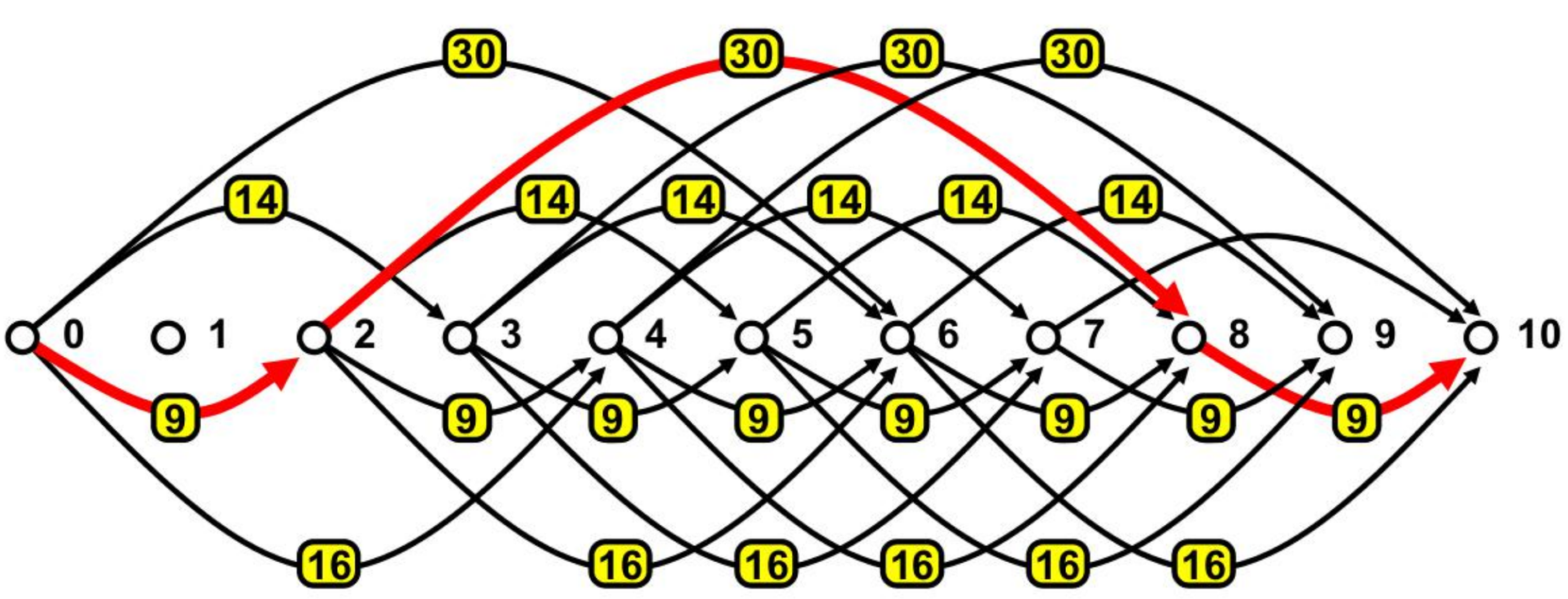


Optimální naplnění kapacity 10 = max(18 + 30, 30 + 16, 32 + 14, 39 + 9) = 48

Neomezená úloha batohu

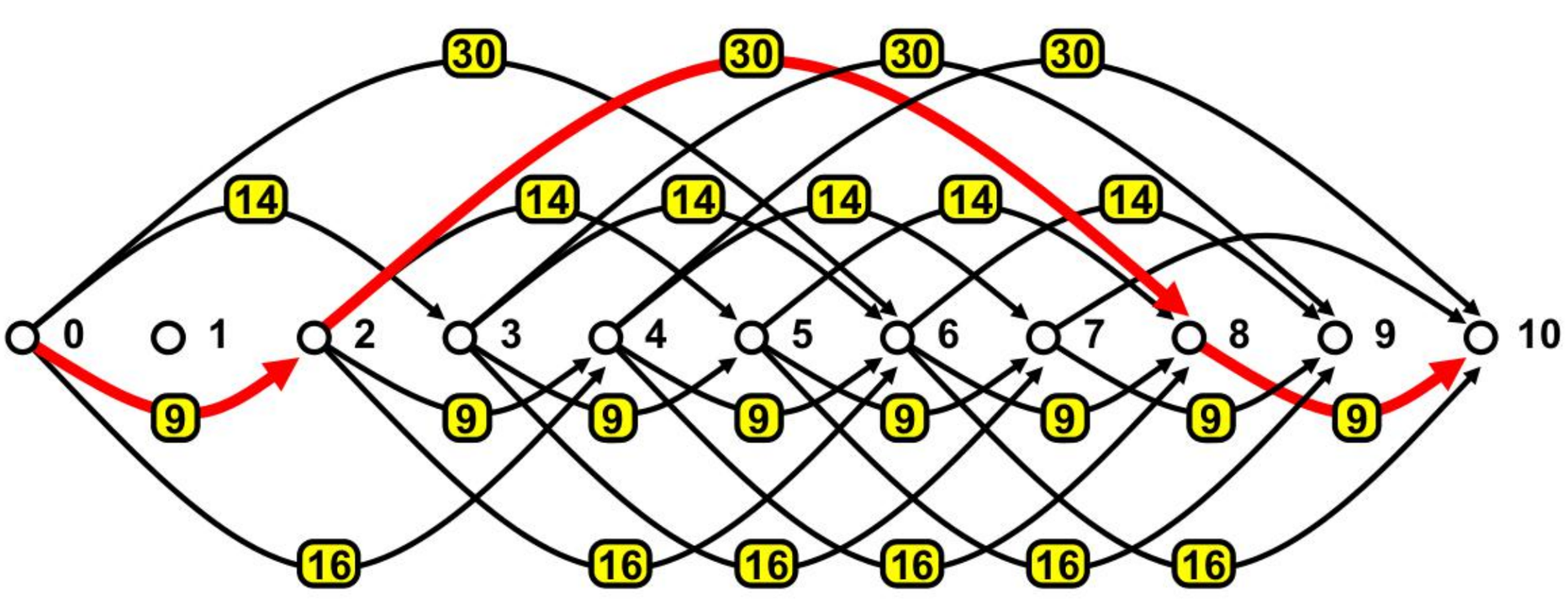
Nejdelší cesta odpovídá optimálnímu naplnění batohu.
Dvě hrany s cenou 9 a jedna hrana s cenou 30, celkem cena = 48.

Batoh optimálně naplníme dvěma předměty s váhou 2 a cenou 9 a jedním předmětem s váhou 6 a cenou 30.



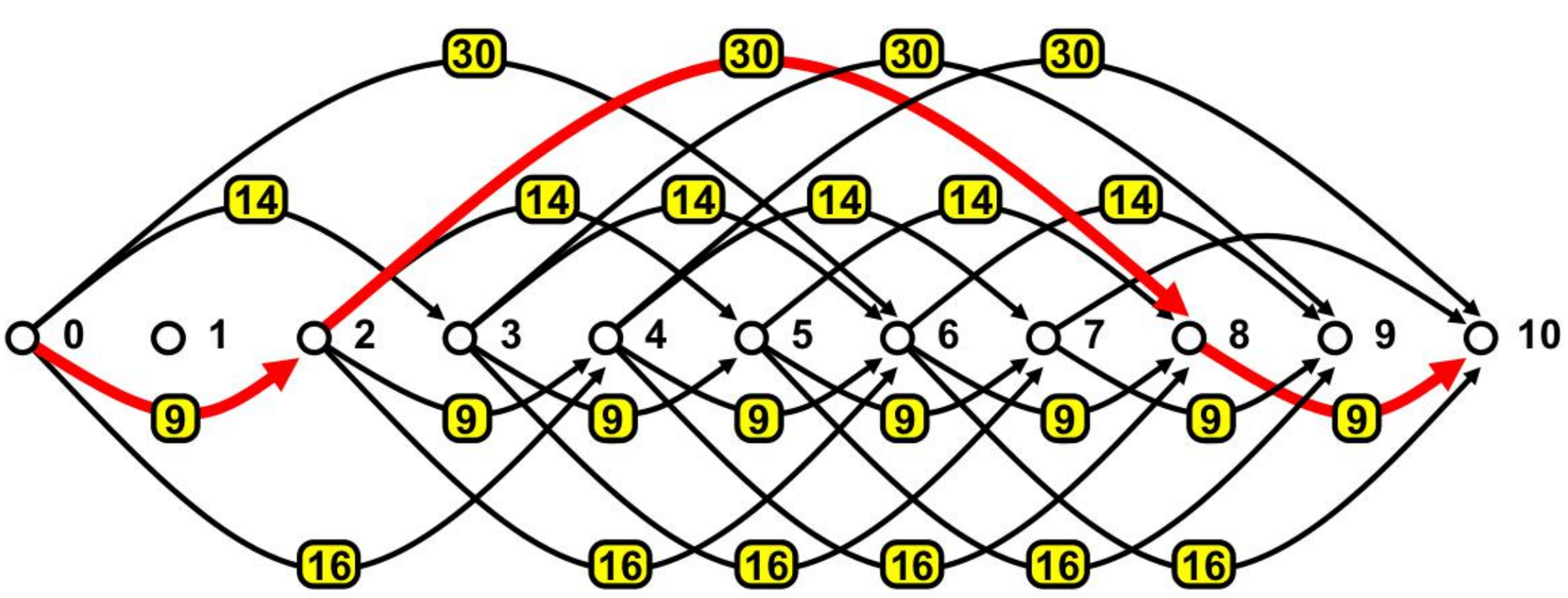
Neomezená úloha batohu -- asymptotická složitost

DAG obsahuje K+1 uzlů a méně než K*N hran.
Má tedy V = Θ(K) uzlů a E = O(K*N) hran.
Asymptotická složitost hledání nejdelší cesty je Θ(V+E),
máme tedy pro neomezenou úlohu batohu
asymptotickou složitost O(K + K*N) = O(K*N).



Neomezená úloha batohu -- asymptotická složitost

DAG obsahuje K+1 uzlů a méně než K*N hran.
Má tedy V = Θ(K) uzlů a E = O(K*N) hran.
Asymptotická složitost hledání nejdelší cesty je Θ(V+E),
máme tedy pro neomezenou úlohu batohu
asymptotickou složitost O(K + K*N) = O(K*N).



Neomezená úloha batohu -- Asymptotická složitost

Zdánlivá nesrovnalost

1. Literatura: NP těžký problém, není znám efektivní algoritmus.
2. ALG OI: DP řeší úlohu v čase v O(N*K), tedy efektivně?

Délka výpočtu DP je lineárně závislá na velikosti kapacity K.

Příklad
Velkou kapacitu 2^{64} lze zadat velmi krátkým zápisem
Kapacita = 18446744073709551616.
N = 3. Položky (váha, cena): (2, 345), (3, 456), (5, 678).
Data lze zapsat do cca 100 bitů < 16 Bytů < "dva longy"
Výpočet pomocí DP potrvá přes 584 roky
za předpokladu, že za 1 sec vyplní 10^9 prvků tabulky.

Délka výpočtu DP je exponenciálně závislá na délce řetězce definujícího kapacitu K.

Neomezená úloha batohu -- Asymptotická složitost

Zdánlivá nesrovnalost

1. Literatura: NP těžký problém, není znám efektivní algoritmus.
2. ALG OI: DP řeší úlohu v čase v O(N*K), tedy efektivně?

Délka výpočtu DP je lineárně závislá na velikosti kapacity K.

Příklad
Velkou kapacitu 2^{64} lze zadat velmi krátkým zápisem
Kapacita = 18446744073709551616.
N = 3. Položky (váha, cena): (2, 345), (3, 456), (5, 678).
Data lze zapsat do cca 100 bitů < 16 Bytů < "dva longy"
Výpočet pomocí DP potrvá přes 584 roky
za předpokladu, že za 1 sec vyplní 10^9 prvků tabulky.

Délka výpočtu DP je exponenciálně závislá na délce řetězce definujícího kapacitu K.

0/1 úloha batohu

Každý předmět lze použít nejvýše 1 krát.

Máme vybrat vhodnou podmnožinu předmětů splňující zadání úlohy. Každé podmnožině lze přiřadit charakteristický vektor z hodnot 0/1 délky N. Pozice ve vektoru odpovídá předmětu, 0 resp. 1 odpovídá nepřítomnosti resp. přítomnosti předmětu v této podmnožině. Binárních vektorů délky N je celkem 2^N, systematické probírání všech možných podmnožin bude mít exponenciální asymptotickou složitost, nehodí se.

DP poskytuje (pro relativně nevelké kapacity) výhodnější postup.

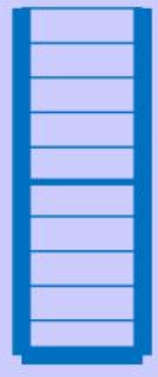
0/1 úloha batohu

Příklad

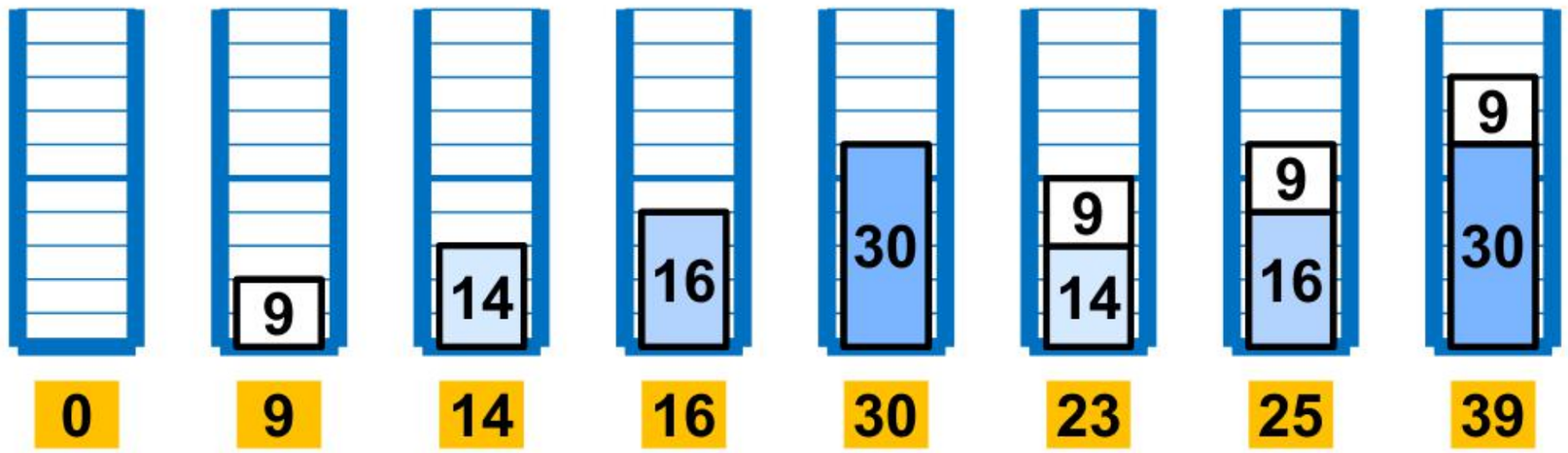
N = 4

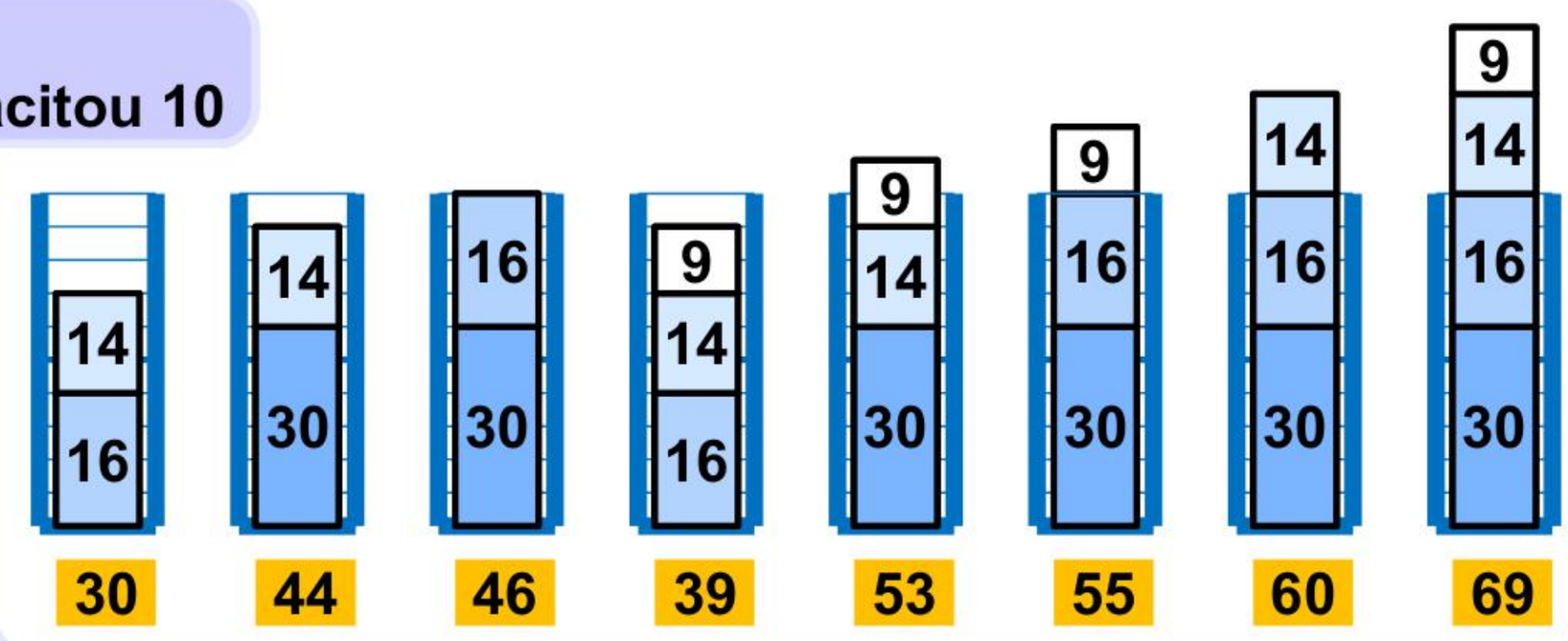
Váha	Cena
2	9
3	14
4	16
6	30

Batoh s kapacitou 10



Všech 16 podmnožin čtyř předmětů a jejich ceny





0/1 úloha batohu -- řešení

Použijeme K+1 batohů, o kapacitách 0, 1, 2, 3, ..., K.
Použijeme N+1 souborů předmětů.

Soubor 0 neobsahuje žádný předmět.
Soubor 1 obsahuje předmět 1.
Soubor 2 obsahuje předměty 1 a 2.
Soubor 3 obsahuje předměty 1, 2, 3.
...
Soubor N obsahuje předměty 1, 2, 3, ..., N.

Na pořadí předmětů nezáleží, je ale zafixované.

Pro každou kapacitu a pro každý soubor budeme řešit
stejnou úlohu metodou DP, v pořadí od menších hodnot k větším.

0/1 úloha batohu -- řešení

Označme symbolem U(x, y) úlohu
se souborem předmětů 1, 2, ..., x a s kapacitou batohu y
a symbolem Opt(x, y) optimální řešení této úlohy.

Pro řešení U(x,y) použijeme optimální řešení úloh U(x-1, _):

Buď do Opt(x, y) zahrneme předmět x nebo jej nezahrneme.
V prvním případě použijeme hodnotu řešení pro batoh
s kapacitou menší o velikost váhy Vx, tedy hodnotu Opt(x-1, y-Vx),
ke které přičteme cenu Cx předmětu x.
V druhém případě beze změny použijeme hodnotu Opt(x-1, y).
Z obou hodnot vybereme tu výhodnější a dostáváme tak:

$$Opt(x, y) = \max(Opt(x-1, y),\ Opt(x-1, y-Vx) + Cx).$$

Dále zřejmě platí Opt(0, y) = Opt(x, 0) = 0, pro x = 0..N, y = 0..K.

0/1 úloha batohu -- řešení

Pro x = 1..N, y = 0..K:

Opt(x, y) = max(Opt(x-1, y), Opt(x-1, y-Vx) + Cx).
Opt(0, y) = Opt(x, 0) = Opt(0, 0) = 0.

Pokud y-Vx < 0, položíme Opt(x, y-Vx) = $-\infty$ (a netabelujeme).

Hodnoty Opt(x,y) tabelujeme ve 2D tabulce velikosti (N+1)×(K+1)
s řádkovým indexem x (předměty)
a sloupcovým indexem y (kapacity menších batohů).

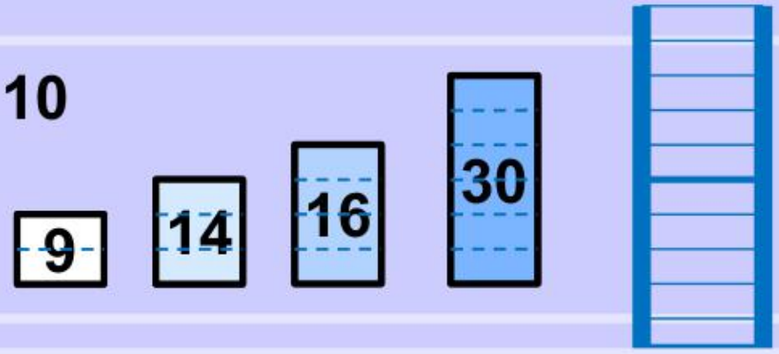
Pro rekonstrukci optimálního řešení použijeme tabulku předchůdců stejné velikosti jako tabulku pro Opt(x, y).
Předchůdce leží vždy v předchozím řádku x-1, stačí registrovat buď pozici y (beze změny) nebo pozici y-Vx (přidán předmět x).

0/1 úloha batohu

Příklad

N = 4 Kapacita = 10

Váha	2	3	4	6
Cena	9	14	16	30



Opt(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	9	9	9	9	9	9	9	9	9
2	0	0	9	14	14	23	23	23	23	23	23
3	0	0	9	14	16	23	25	30	30	39	39
4	0	0	9	14	16	23	30	30	39	44	46

Pred(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4

0/1 úloha batohu -- řešení

Pro x = 1..N, y = 0..K:

Opt(x, y) = max(Opt(x-1, y), Opt(x-1, y-Vx) + Cx).
Opt(0, y) = Opt(x, 0) = Opt(0, 0) = 0.

Pokud y-Vx < 0, položíme Opt(x, y-Vx) = $-\infty$ (a netabelujeme).

Hodnoty Opt(x,y) tabelujeme ve 2D tabulce velikosti (N+1)×(K+1)
s řádkovým indexem x (předměty)
a sloupcovým indexem y (kapacity menších batohů).

Pro rekonstrukci optimálního řešení použijeme tabulku předchůdců stejné velikosti jako tabulku pro Opt(x, y).
Předchůdce leží vždy v předchozím řádku x-1, stačí registrovat buď pozici y (beze změny) nebo pozici y-Vx (přidán předmět x).

0/1 úloha batohu -- řešení

Označme symbolem U(x, y) úlohu
se souborem předmětů 1, 2, ..., x a s kapacitou batohu y
a symbolem Opt(x, y) optimální řešení této úlohy.

Pro řešení U(x,y) použijeme optimální řešení úloh U(x-1, _):

Buď do Opt(x, y) zahrneme předmět x nebo jej nezahrneme.
V prvním případě použijeme hodnotu řešení pro batoh
s kapacitou menší o velikost váhy Vx, tedy hodnotu Opt(x-1, y-Vx),
ke které přičteme cenu Cx předmětu x.
V druhém případě beze změny použijeme hodnotu Opt(x-1, y).
Z obou hodnot vybereme tu výhodnější a dostáváme tak:

$$Opt(x, y) = \max(Opt(x-1, y),\ Opt(x-1, y-Vx) + Cx).$$

Dále zřejmě platí Opt(0, y) = Opt(x, 0) = 0, pro x = 0..N, y = 0..K.

0/1 úloha batohu -- řešení

Označme symbolem U(x, y) úlohu
se souborem předmětů 1, 2, ..., x a s kapacitou batohu y
a symbolem Opt(x, y) optimální řešení této úlohy.

Pro řešení U(x,y) použijeme optimální řešení úloh U(x-1, _):

Buď do Opt(x, y) zahrneme předmět x nebo jej nezahrneme.
V prvním případě použijeme hodnotu řešení pro batoh
s kapacitou menší o velikost váhy Vx, tedy hodnotu Opt(x-1, y-Vx),
ke které přičteme cenu Cx předmětu x.
V druhém případě beze změny použijeme hodnotu Opt(x-1, y).
Z obou hodnot vybereme tu výhodnější a dostáváme tak:

$$Opt(x, y) = \max(Opt(x-1, y),\ Opt(x-1, y-Vx) + Cx).$$

Dále zřejmě platí $Opt(0, y) = Opt(x, 0) = 0$, pro $x = 0..N$, $y = 0..K$.

0/1 úloha batohu -- řešení

Pro x = 1..N, y = 0..K:

Opt(x, y) = max(Opt(x-1, y), Opt(x-1, y-Vx) + Cx).
Opt(0, y) = Opt(x, 0) = Opt(0, 0) = 0.

Pokud y-Vx < 0, položíme Opt(x, y-Vx) = $-\infty$ (a netabelujeme).

Hodnoty Opt(x,y) tabelujeme ve 2D tabulce velikosti (N+1)×(K+1)
s řádkovým indexem x (předměty)
a sloupcovým indexem y (kapacity menších batohů).

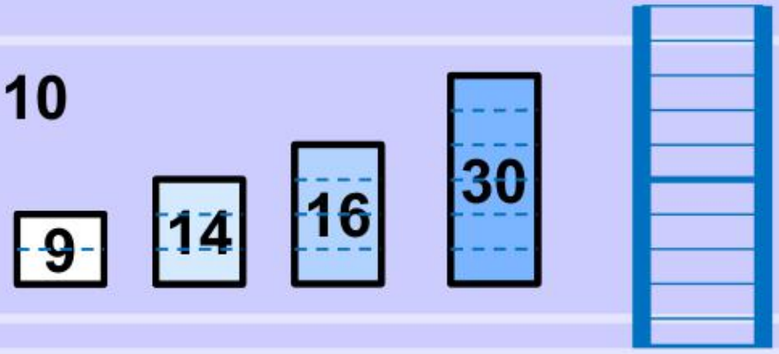
Pro rekonstrukci optimálního řešení použijeme tabulku předchůdců stejné velikosti jako tabulku pro Opt(x, y).
Předchůdce leží vždy v předchozím řádku x-1, stačí registrovat buď pozici y (beze změny) nebo pozici y-Vx (přidán předmět x).

0/1 úloha batohu

Příklad

N = 4 Kapacita = 10

Váha	2	3	4	6
Cena	9	14	16	30



Opt(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	9	9	9	9	9	9	9	9	9
2	0	0	9	14	14	23	23	23	23	23	23
3	0	0	9	14	16	23	25	30	30	39	39
4	0	0	9	14	16	23	30	30	39	44	46

Pred(x, y)

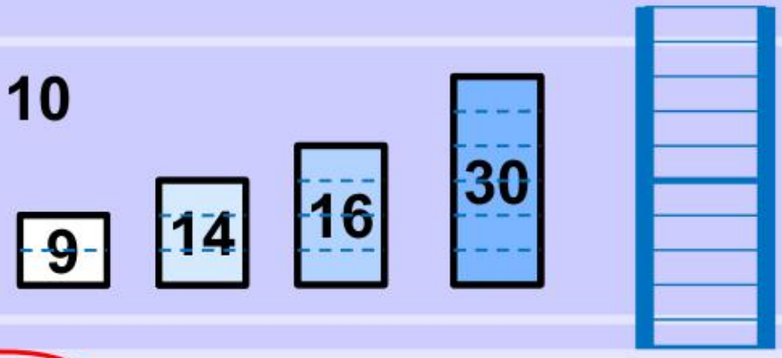
	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4

0/1 úloha batohu

Příklad

N = 4 Kapacita = 10

Váha	2	3	4	6
Cena	9	14	16	30



Opt(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	9	9	9	9	9	9	9	9	9
2	0	0	9	14	14	23	23	23	23	23	23
3	0	0	9	14	16	23	25	30	30	39	39
4	0	0	9	14	16	23	30	30	39	44	46

Pred(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4

0/1 úloha batohu

Vyjádření jako optimální cesty v DAG

Uzly DAG budou jednotlivé hodnoty Opt(x, y), x = 0..N, y = 0..K,
celkem bude mít DAG (N+1)*(K+1) uzlů.
Do uzlu Opt(x, y) povede hrana

-- Opt(x-1, y) --> Opt(x, y)
ohodnocená 0 (žádný přidaný předmět),

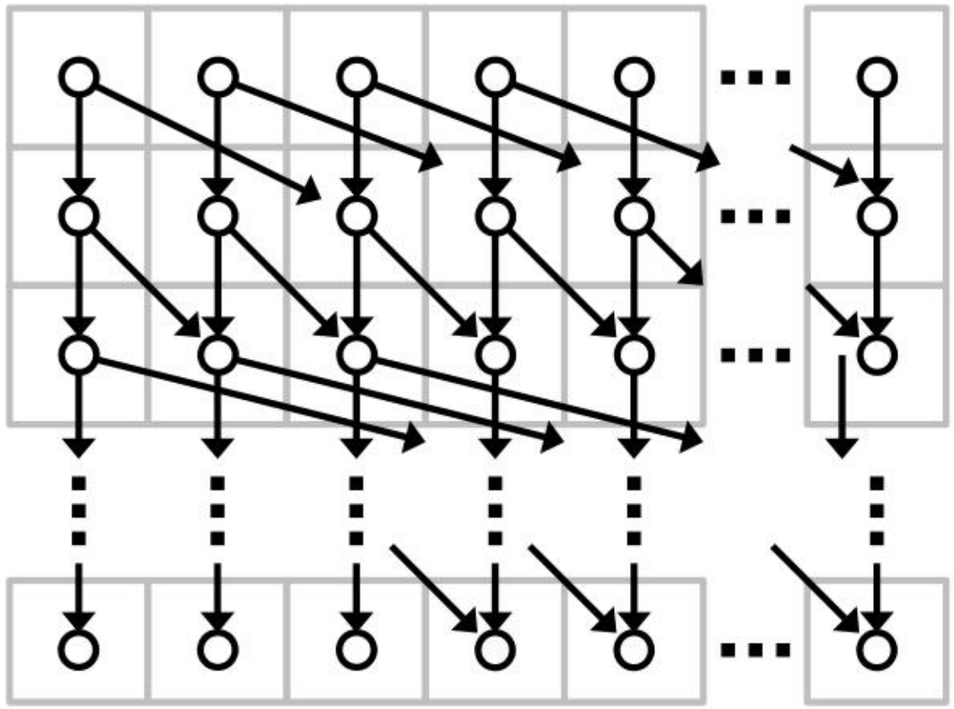
-- a pokud $y-Vx \geq 0$, také hrana
Opt(x-1, y-Vx) --> Opt(x, y)
ohodnocená cenou Cx (cenou přidaného předmětu x).

V takto zkonstruovaném DAG hledáme nejdelší (= nejcennější)
cestu standardní DP metodou.

Jaké je topologické uspořádání tohoto DAG?

0/1 úloha batohu - topologické uspořádání DAG

DAG můžeme uvažovat nakreslený formálně do DP tabulky, přičemž uzel Opt(x, y) leží v buňce s indexy x a y. Pak hrany DAG vedou vždy pouze z předchozího řádku do následujícího řádku. Pokud tento DAG procházíme shora po řádcích, to jest ve stejném pořadí, v němž vyplňujeme DP tabulku, respektujeme jeho topologické uspořádání.



V tomto případě není nutno uzly DAG v topologickém uspořádání uvažovat v jedné přímce, "tabulkové" uspořádání je přehlednější.

0/1 úloha batohu

Vyjádření jako optimální cesty v DAG

Uzly DAG budou jednotlivé hodnoty Opt(x, y), x = 0..N, y = 0..K,
celkem bude mít DAG (N+1)*(K+1) uzlů.
Do uzlu Opt(x, y) povede hrana

-- Opt(x-1, y) --> Opt(x, y)
ohodnocená 0 (žádný přidaný předmět),

-- a pokud $y-Vx \geq 0$, také hrana
Opt(x-1, y-Vx) --> Opt(x, y)
ohodnocená cenou Cx (cenou přidaného předmětu x).

V takto zkonstruovaném DAG hledáme nejdelší (= nejcennější)
cestu standardní DP metodou.

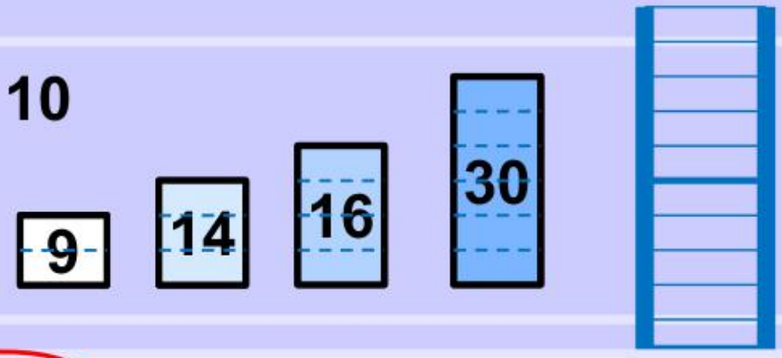
Jaké je topologické uspořádání tohoto DAG?

0/1 úloha batohu

Příklad

N = 4 Kapacita = 10

Váha	2	3	4	6
Cena	9	14	16	30



Opt(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	9	9	9	9	9	9	9	9	9
2	0	0	9	14	14	23	23	23	23	23	23
3	0	0	9	14	16	23	25	30	30	39	39
4	0	0	9	14	16	23	30	30	39	44	46

Pred(x, y)

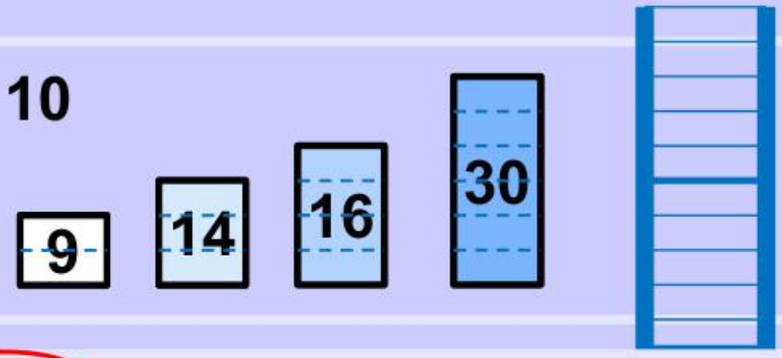
	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4

0/1 úloha batohu

Příklad

N = 4 Kapacita = 10

Váha	2	3	4	6
Cena	9	14	16	30



Opt(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	9	9	9	9	9	9	9	9	9
2	0	0	9	14	14	23	23	23	23	23	23
3	0	0	9	14	16	23	25	30	30	39	39
4	0	0	9	14	16	23	30	30	39	44	46

Pred(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4

0/1 úloha batohu

Vyjádření jako optimální cesty v DAG

Uzly DAG budou jednotlivé hodnoty Opt(x, y), x = 0..N, y = 0..K,
celkem bude mít DAG (N+1)*(K+1) uzlů.
Do uzlu Opt(x, y) povede hrana

-- Opt(x-1, y) --> Opt(x, y)
ohodnocená 0 (žádný přidaný předmět),

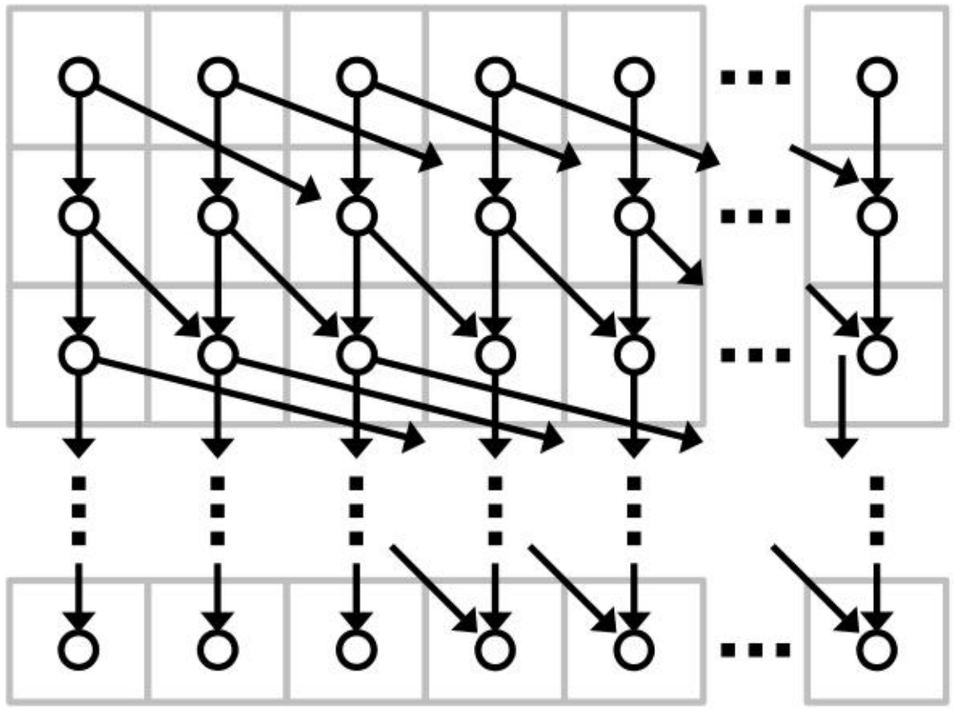
-- a pokud $y-Vx \geq 0$, také hrana
Opt(x-1, y-Vx) --> Opt(x, y)
ohodnocená cenou Cx (cenou přidaného předmětu x).

V takto zkonstruovaném DAG hledáme nejdelší (= nejcennější)
cestu standardní DP metodou.

Jaké je topologické uspořádání tohoto DAG?

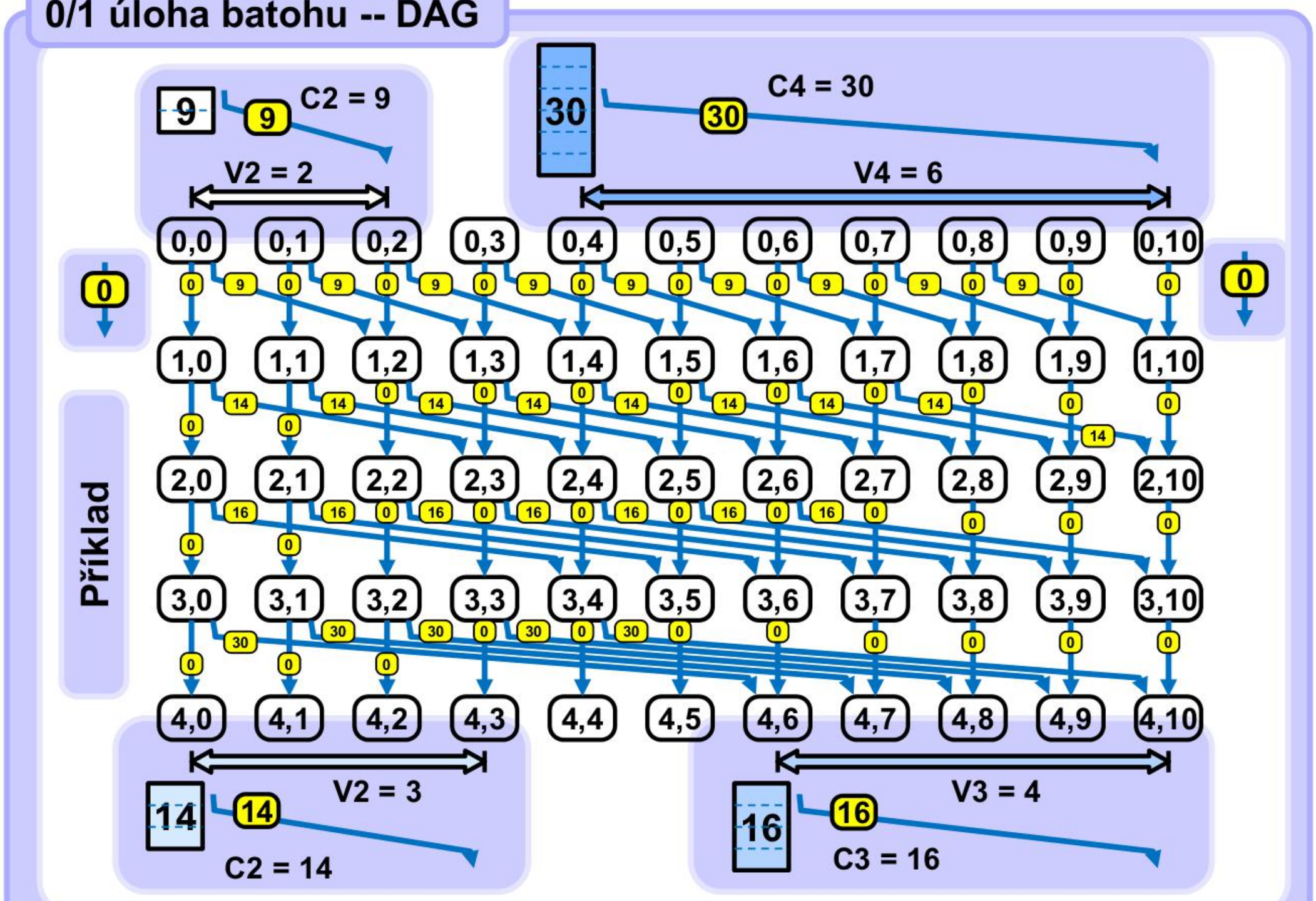
0/1 úloha batohu - topologické uspořádání DAG

DAG můžeme uvažovat nakreslený formálně do DP tabulky, přičemž uzel Opt(x, y) leží v buňce s indexy x a y. Pak hrany DAG vedou vždy pouze z předchozího řádku do následujícího řádku. Pokud tento DAG procházíme shora po řádcích, to jest ve stejném pořadí, v němž vyplňujeme DP tabulku, respektujeme jeho topologické uspořádání.



V tomto případě není nutno uzly DAG v topologickém uspořádání uvažovat v jedné přímce, "tabulkové" uspořádání je přehlednější.

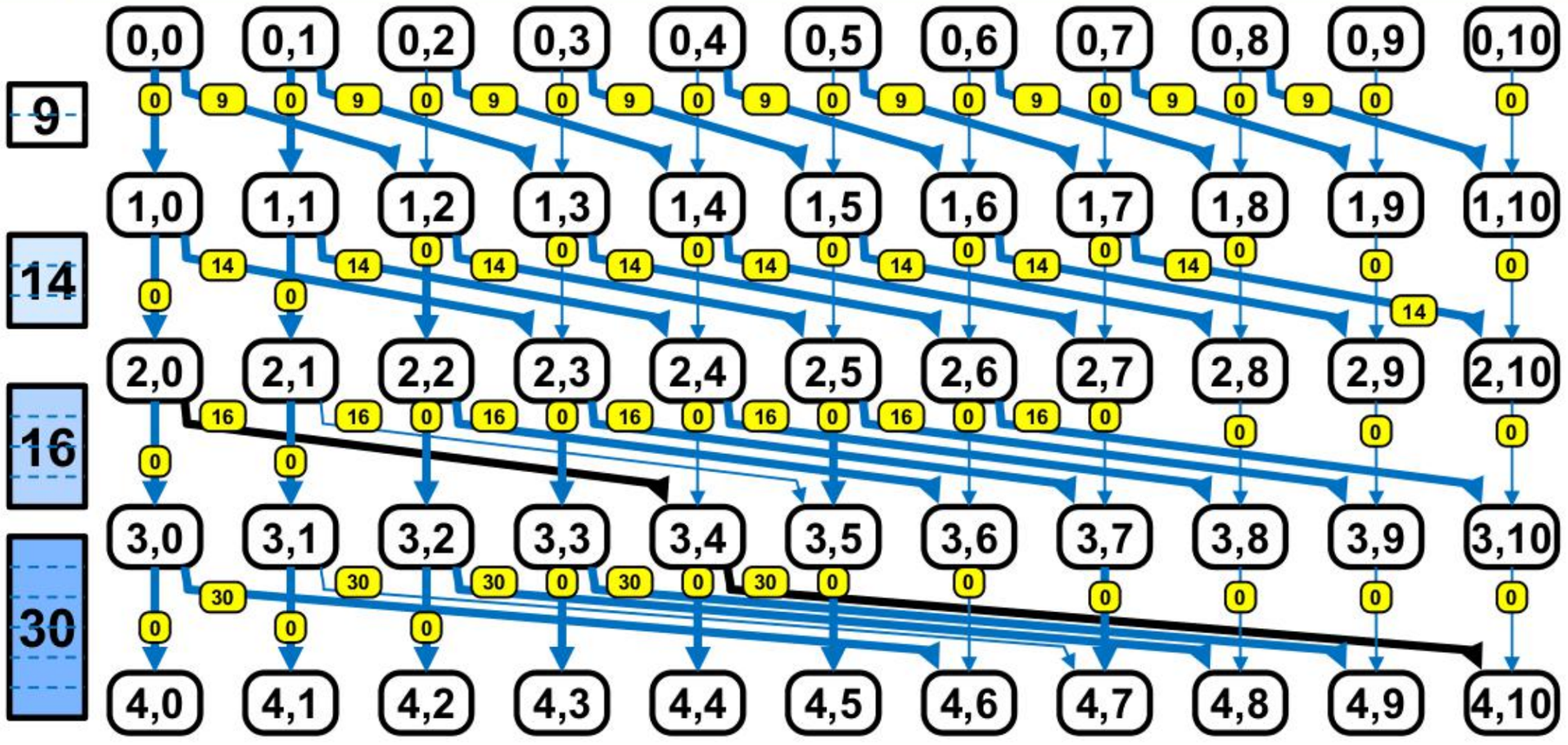
0/1 úloha batohu -- DAG



0/1 úloha batohu -- rekonstrukce optimálního řešení pomocí tabulky předchůdců

Pred(x, y)

	0	1	2	3	4	5	6	7	8	9	10
0	--	--	--	--	--	--	--	--	--	--	--
1	0	1	0	1	2	3	4	5	6	7	8
2	0	1	2	0	1	2	3	4	5	6	7
3	0	1	2	3	0	5	2	3	4	5	6
4	0	1	2	3	4	5	0	7	2	3	4



0/1 úloha batohu

Asymptotická složitost

Tabulka ... Velikost ... (N+1)*(K+1) ∈ Θ(N*K)
Vyplnění jedné buňky ... Θ(1)
Vyplnění tabulky ... Θ(N*K*1) = Θ(N*K).
Rekonstrukce optimálního řešení Θ(N).
Celkem ... Θ(N*K + N) = Θ(N*K).

DAG .... Uzlů ... (N+1)*(K+1) ∈ Θ(N*K).
Hran ... nejvýše 2*(N+1)*(K+1), tj ∈ O(N*K).
Nalezení optimální cesty ... Θ(uzlů+hran) = Θ(N*K).

Řešení obou variant úlohy batohu, neomezené i 0/1, má asymptotickou složitost Θ(N*K).

Přitom zároveň platí:

Asymptotická složitost DP řešení je exponenciální vzhledem k délce řetězce definujícího kapacitu K.