

DSA

Různé algoritmy mají různou složitost

Algoritmus a program není totéž

DSA

Dynamické programování

Myšlenka dynamického programování

Definice
funkce

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Otázka

$f(10,10) = ?$

Program

```
int f(int x, int y) {  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return (2* f(x, y-1) + f(x-1, y));  
}
```

```
print( f(10,10) );
```

Odpověď

$f(10,10) = 127\,574\,017$ 😊

Myšlenka dynamického programování

Jednoduchá analýza

```
int count = 0;  
  
public static int f(int x, int y) {  
    count++;  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return (2* f(x, y-1) + f(x-1,y));  
}
```

```
xyz = f(10,10);  
print(count);
```

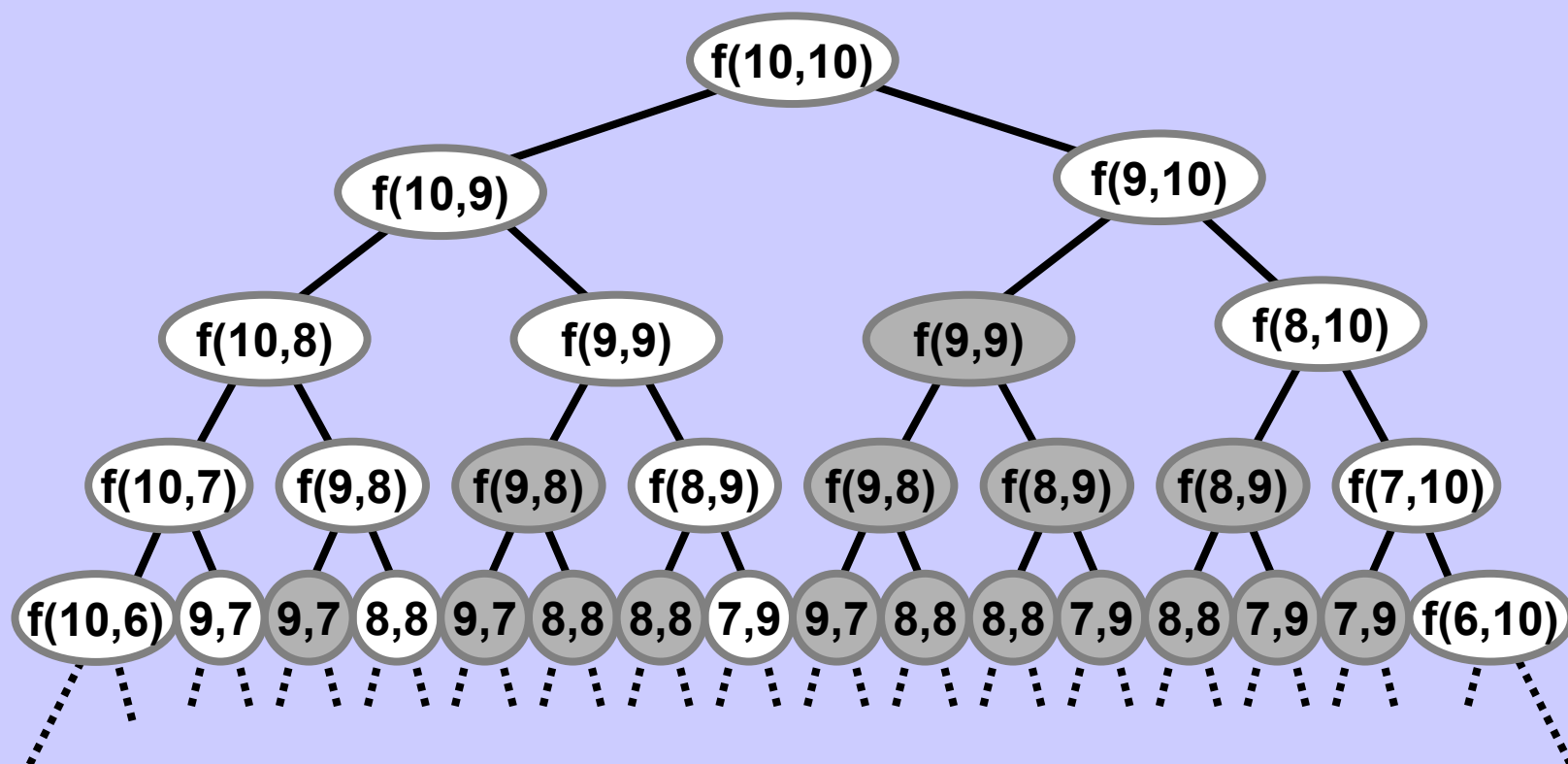
Výsledek analýzy

count = 369 511



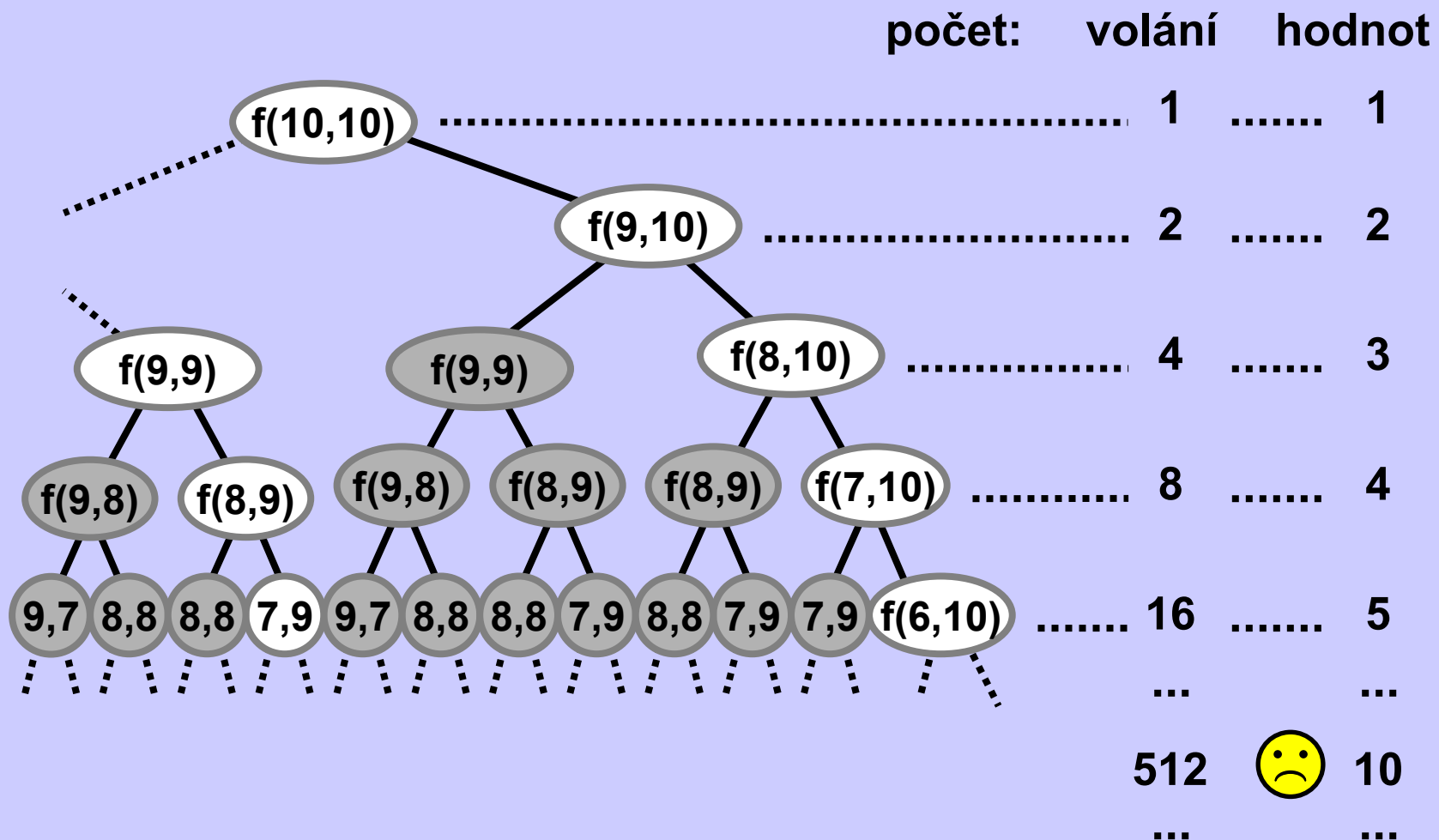
Myšlenka dynamického programování

Detailnější analýza – strom rekurzivního volání



Myšlenka dynamického programování

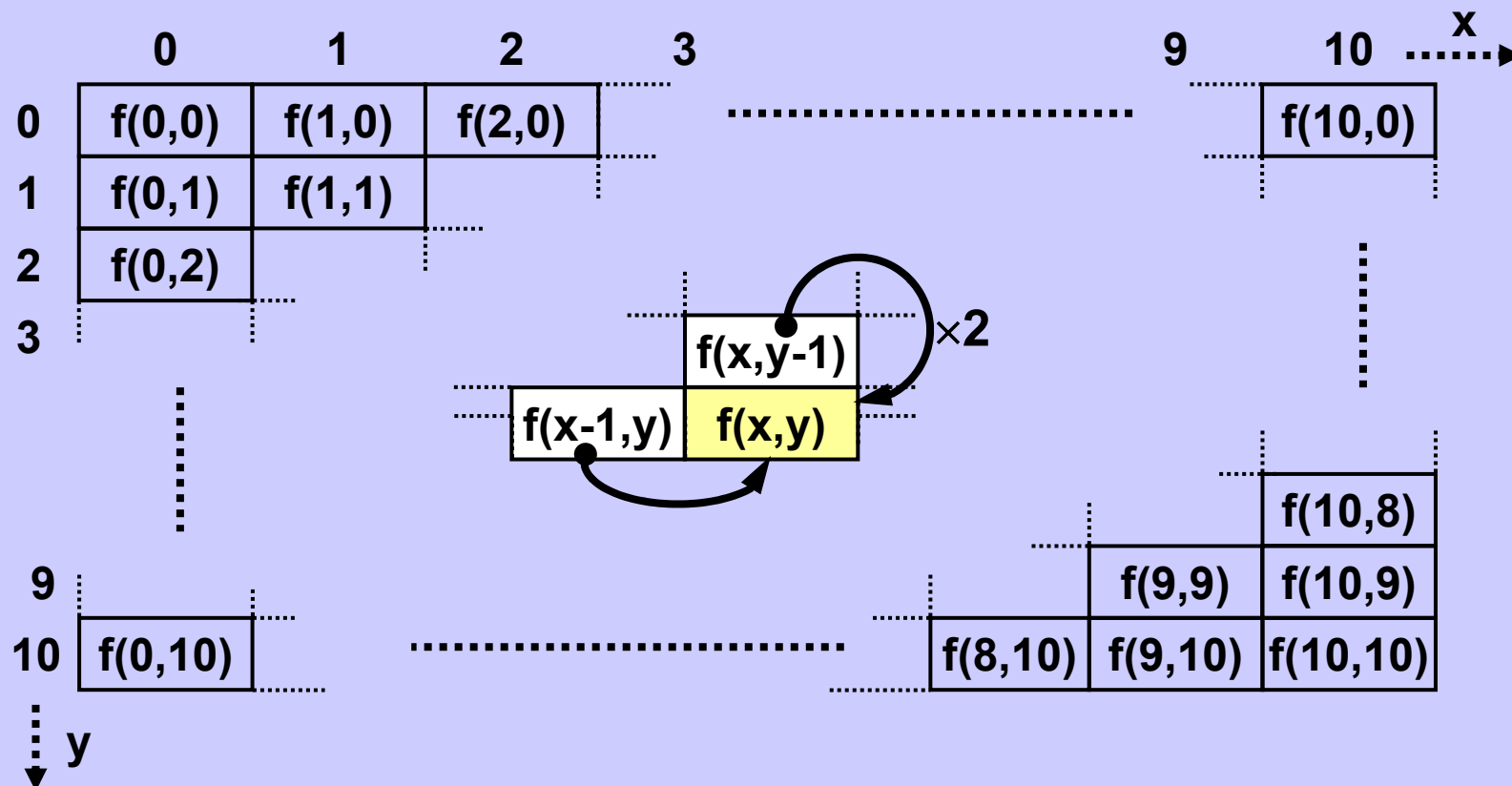
Detailnější analýza pokračuje – efektivita rekurzivního volání



Algoritmus a program není totéž

Myšlenka dynamického programování

$$f(x,y) = \begin{cases} 1 & (x = 0) \parallel (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \& (y > 0) \end{cases}$$



Myšlenka dynamického programování

Všechny hodnoty se předpočítají

```
static int dynArr [N+1][N+1];

void fillDynArr() {
    int xy, x, y;
    for (xy = 0; xy <= N; xy++)
        dynArr[0][xy] = dynArr[xy][0] = 1;

    for (y = 1; y <= N; y++)
        for (x = 1; x <= N; x++)
            dynArr[y][x] = 2*dynArr[y-1][x] + dynArr[y][x-1];
}
```

Volání funkce

```
int f(int x,int y) {
    return dynArr[y][x];
}
```

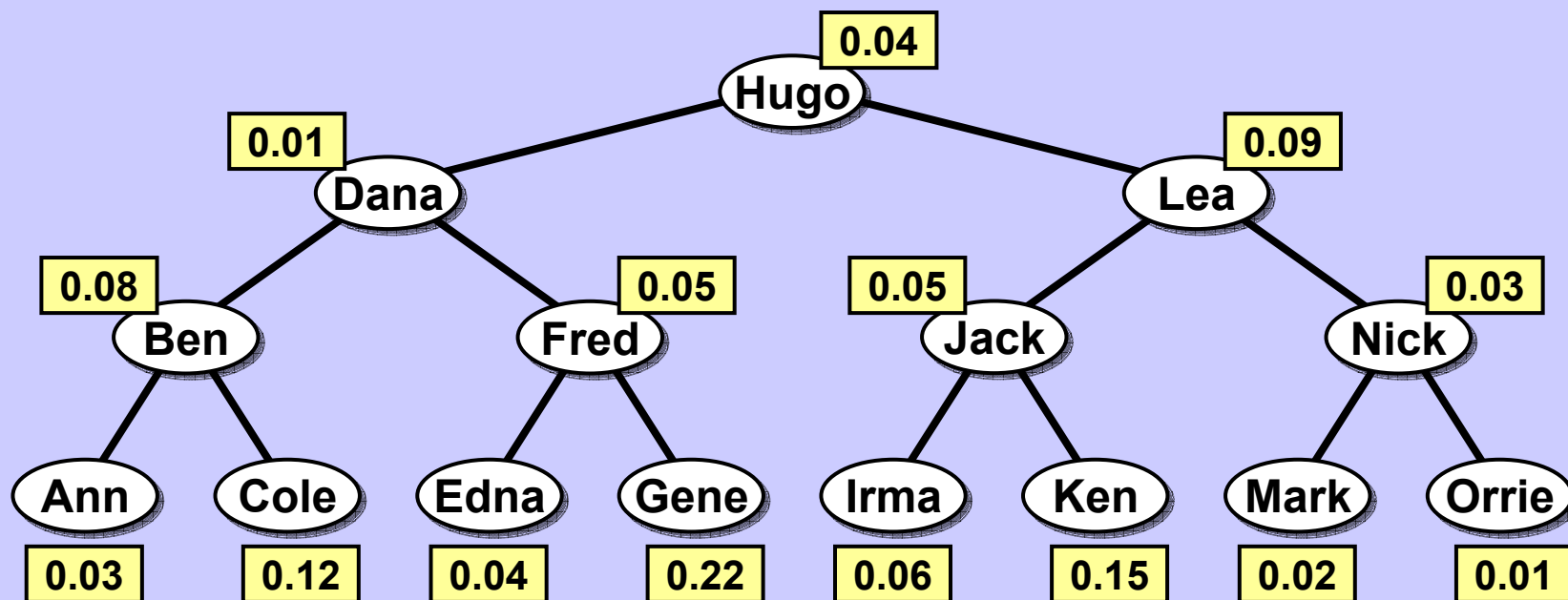
DSA

Dynamické programování

Optimální binární vyhledávací strom

Optimální binární vyhledávací strom

Vyvážený, ale ne optimální

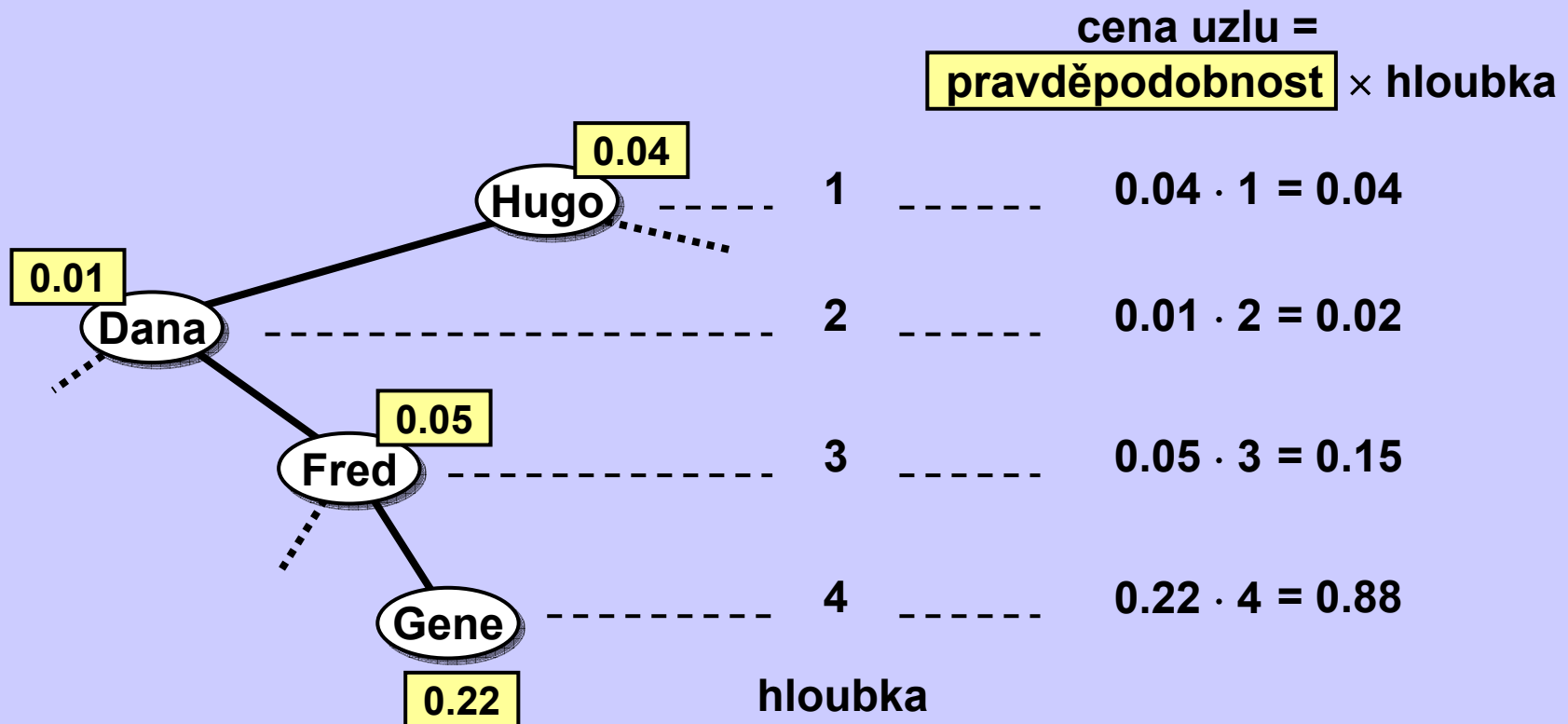


Klíč

Pravděpodobnost dotazu

Optimální binární vyhledávací strom

Cena jednotlivých uzlů v BVS



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

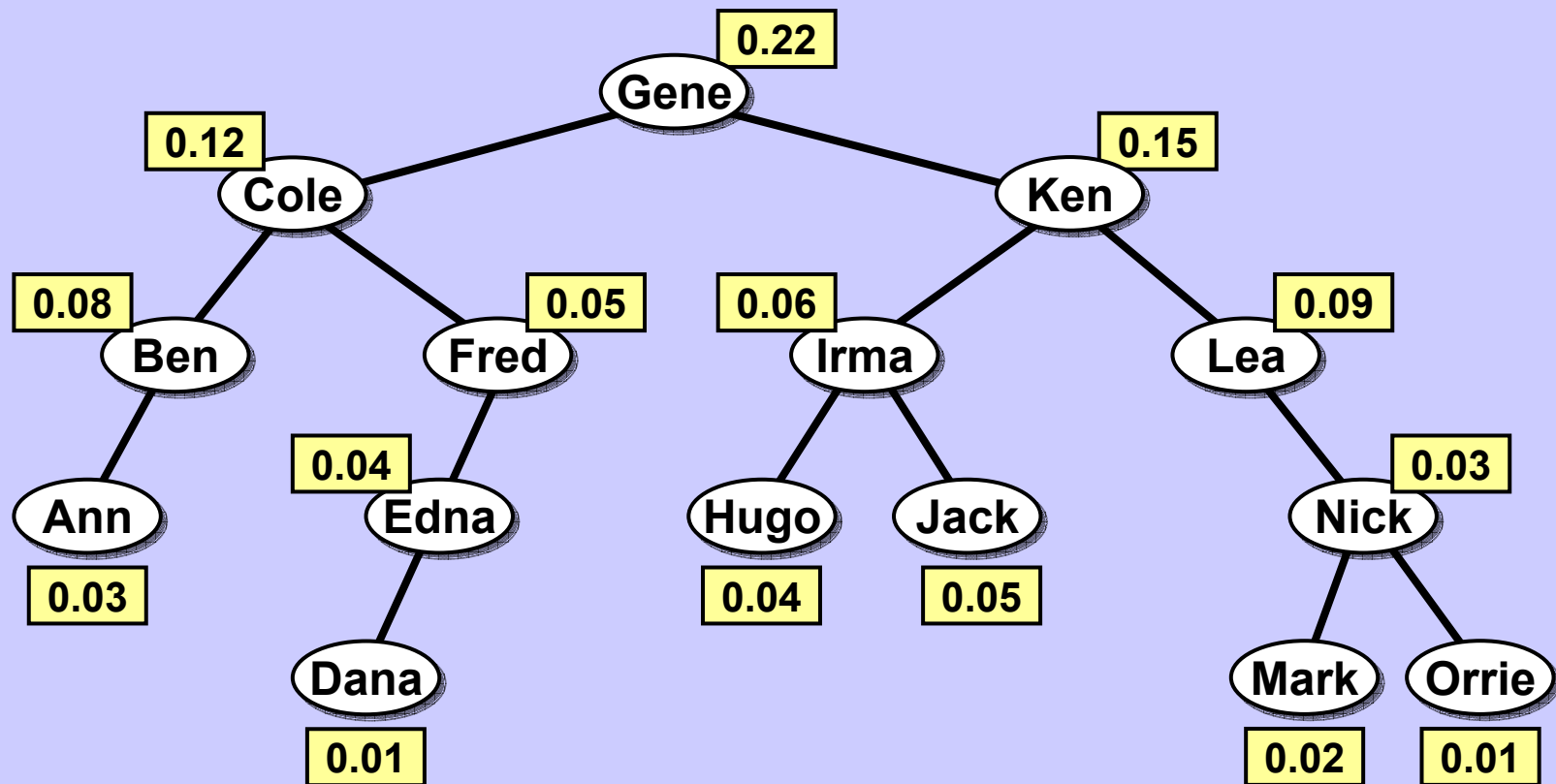
Cena vyváženého stromu

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

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

Optimální BVS

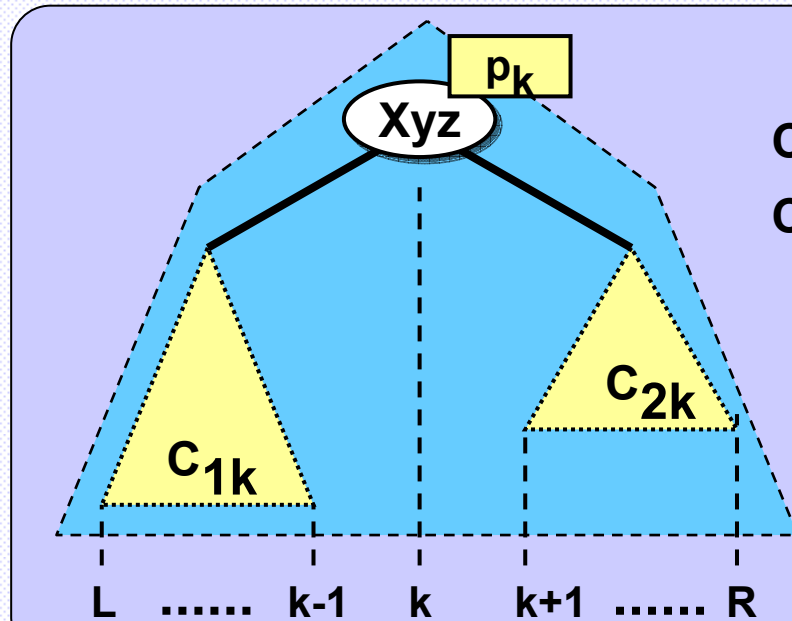
Struktura optimálního BVS s danými pravděpodobnostmi



Cena optimálního BVS

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

Výpočet ceny optimálního BVS



C_{1k} cena levého podstromu uzlu k

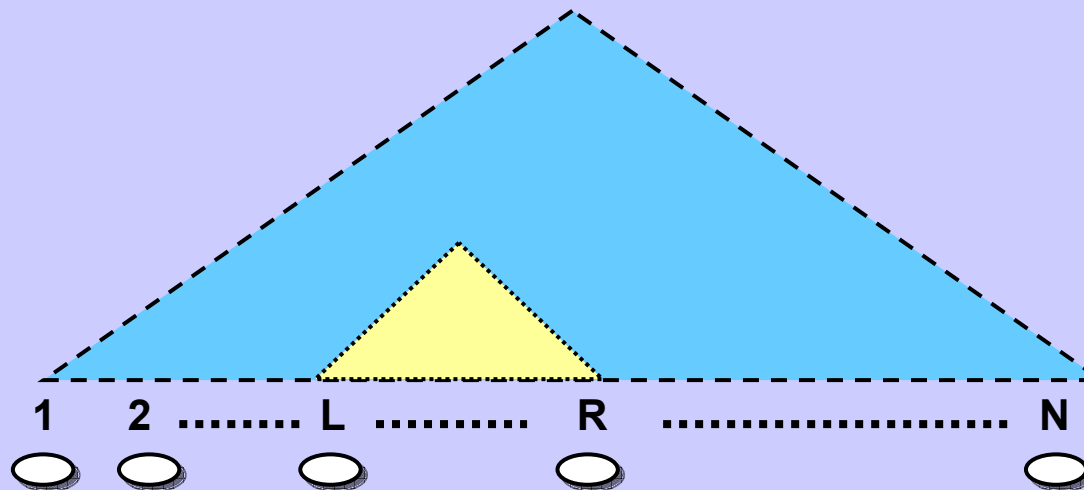
C_{2k} Cena pravého podstromu uzlu k

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.

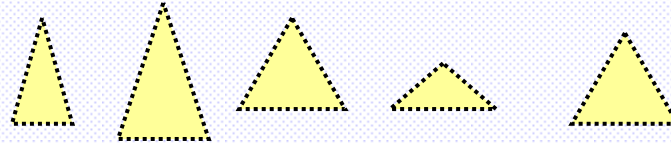
**Velikost stromu
= poč. uzlů
= $R-L+1$**

Máme	N	optimálních podstromů velikosti	1
	$N-1$		2
	$N-2$		3
	\vdots		\vdots
	1	podstrom	N

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

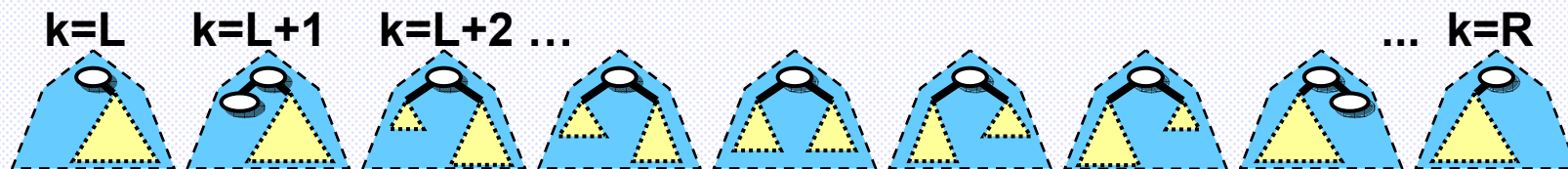
Minimalizace ceny BVS

Idea rekurzivního řešení:



1. Předpoklad : Všechny menší optimální stromy jsou známy.

2. Zkus: $k = L, L+1, L+2, \dots, R$



3. Zaregistruj index k , který minimalizuje cenu, tj. zaregistruj 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 $L, L+1, L+2, \dots, R-1, R$

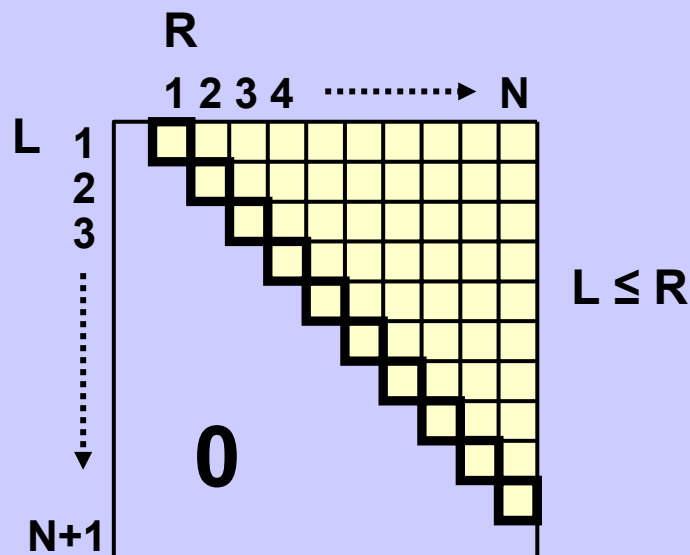
$$\begin{aligned}
 C(L,R) &= \min_{L \leq k \leq R} \left\{ C(L, k-1) + \sum_{i=L}^{k-1} p_i + C(k+1,R) + \sum_{i=k+1}^R p_i + p_k \right\} = \\
 &= \min_{L \leq k \leq R} \left\{ C(L, k-1) + C(k+1,R) + \sum_{i=L}^R p_i \right\} = \\
 (*) &= \min_{L \leq k \leq R} \left\{ C(L, k-1) + C(k+1,R) \right\} + \sum_{i=L}^R p_i
 \end{aligned}$$

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

Datové struktury pro výpočet optimálního BVS

Ceny optimálních podstromů

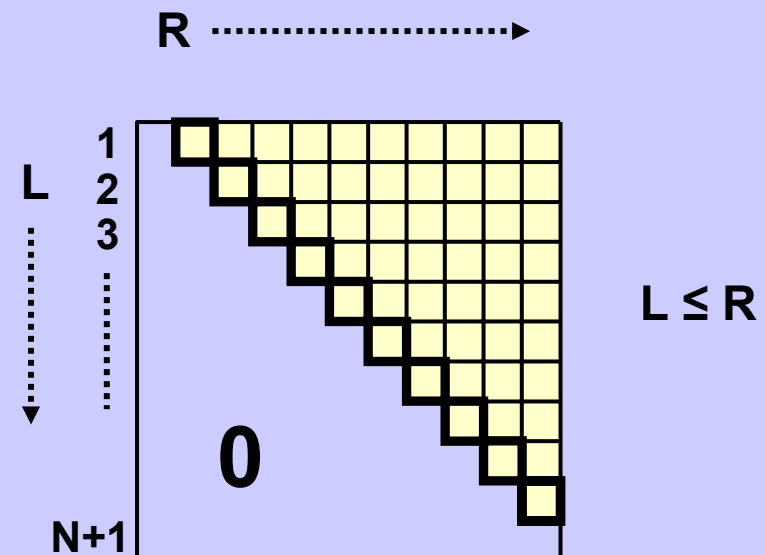
pole $C[L][R]$ ($L \leq R$)



diagonála ... $L = R$

Kořeny optimálních podstromů

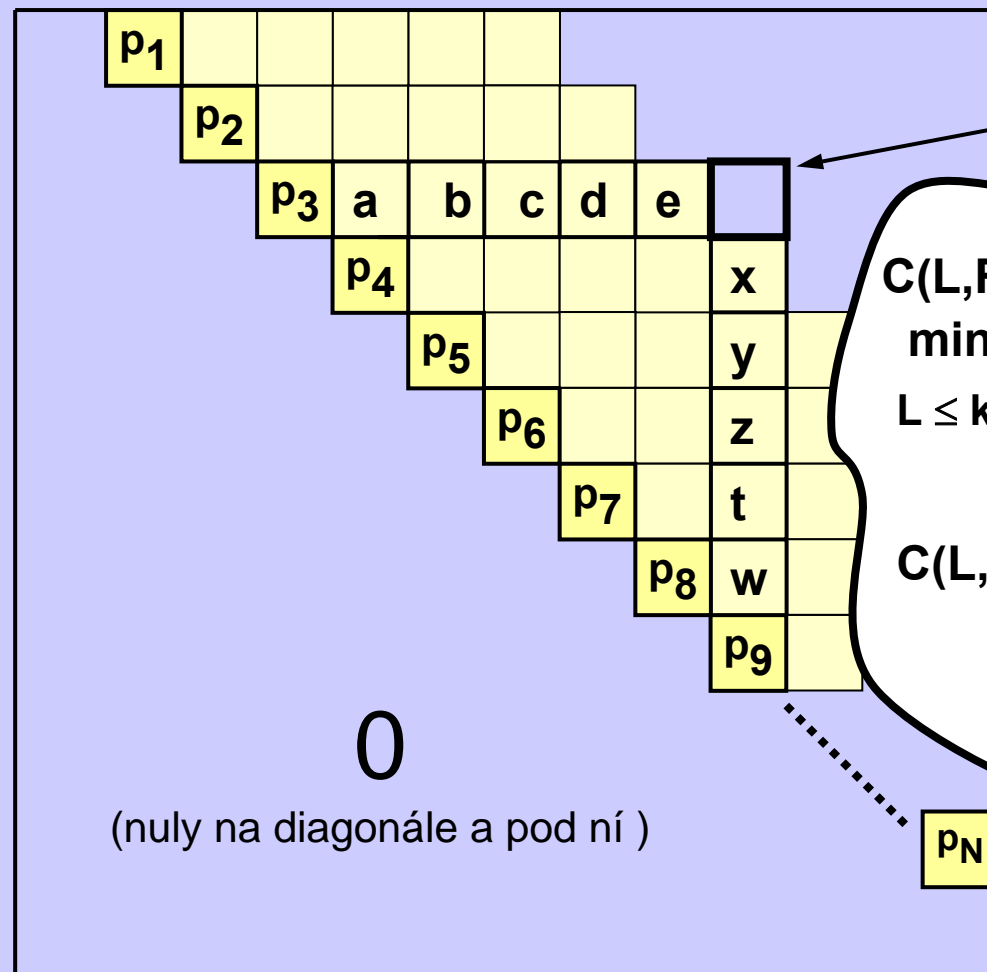
pole $roots[L][R]$ ($L \leq R$)



diagonála ... $L = R$

Výpočet optimálního BVS

Cena konkrétního optimálního podstromu



L=3, R=9

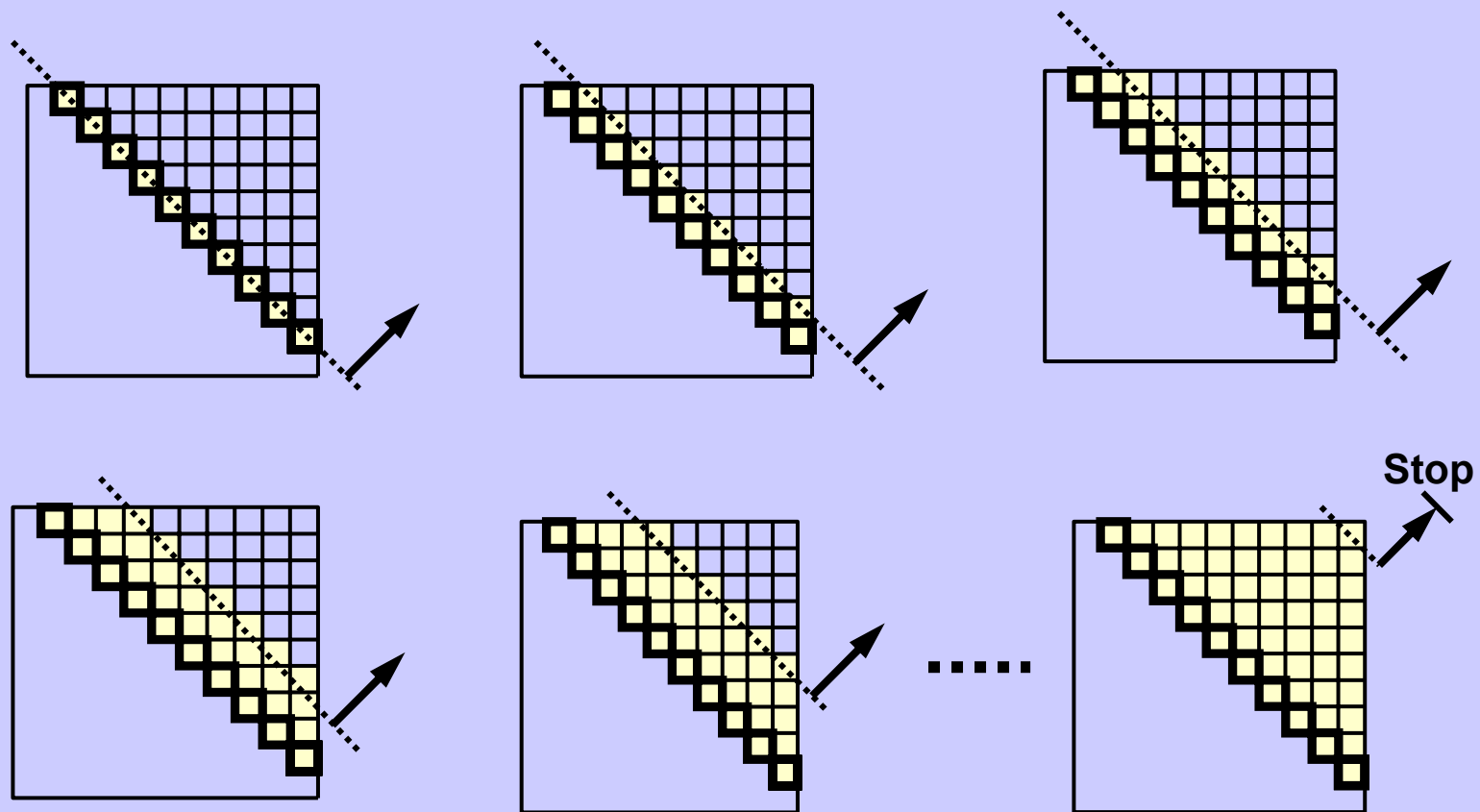
$$C(L,R) = \min_{L \leq k \leq R} \{ C(L, k-1) + C(k+1, R) \} + \sum_{i=L}^R p_i$$

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

Výpočet optimálního BVS

Strategie dynamického programování

– nejprve se zpracují nejmenší podstromy, pak větší, atd...



Algoritmus a program není totéž

Výpočet optimálního BVS

Výpočet polí cen a kořenů

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

    // size = 1
    for (int i=0; i<=N; i++) {
        C[i][i] = pravděpodobnost[i]; R[i][i] = i;
    }
    // size > 1
    for (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 pole kořenů

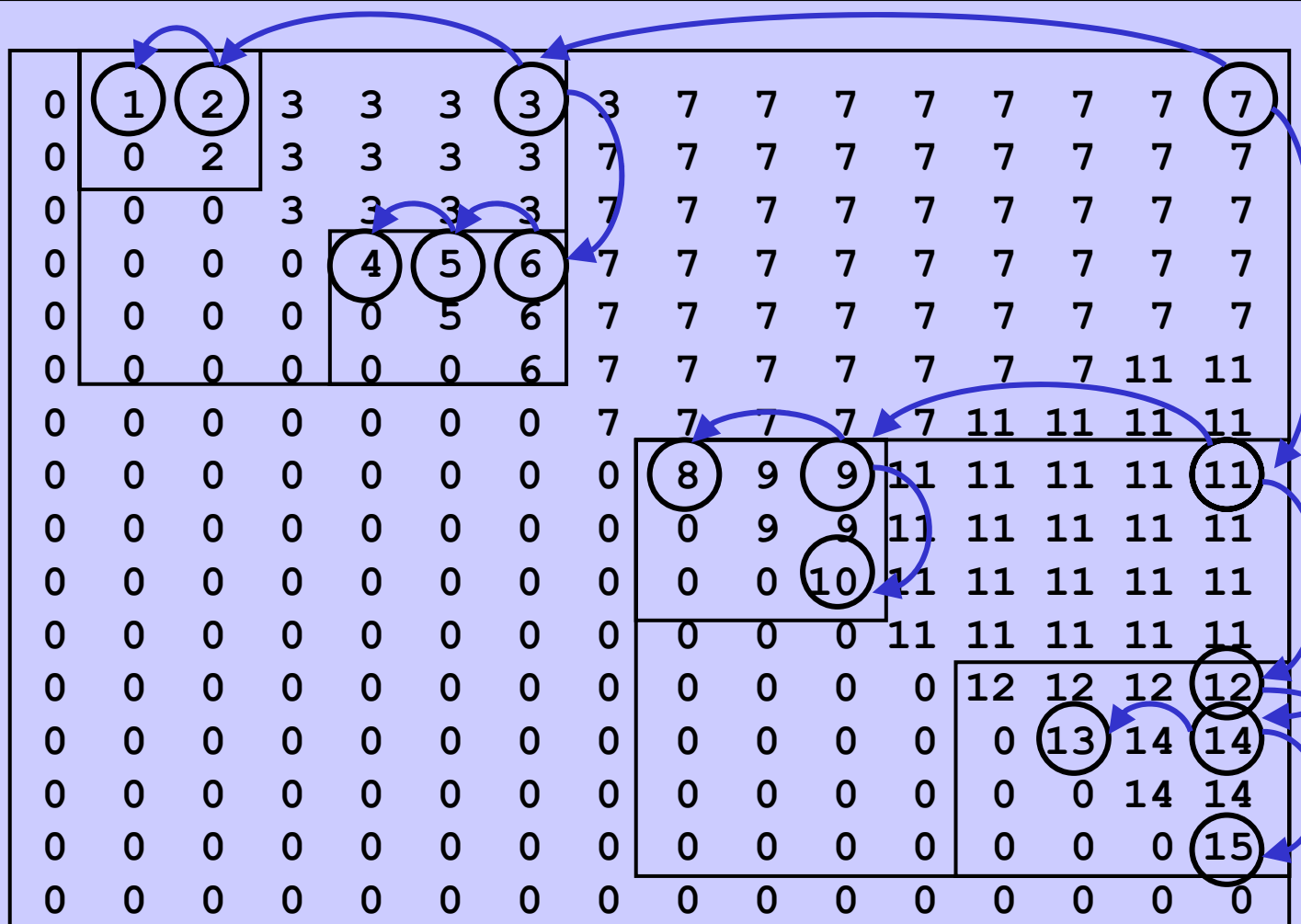
```
void buildTree(int L, int R) {  
    int key;  
    if (R < L) return;  
    key = roots[L][R];  
    insert(root, key);    // standard BST insert  
    buildTree( L, key-1);  
    buildTree( key+1, R);  
}
```

Volání funkce buildTree vybuduje optimální strom:

```
buildTree(1,N);
```

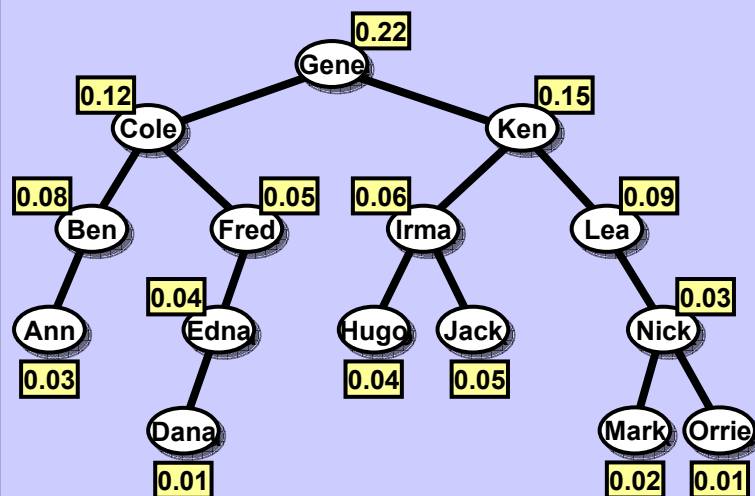
Výpočet optimálního BVS

Kořeny optimálních podstromů

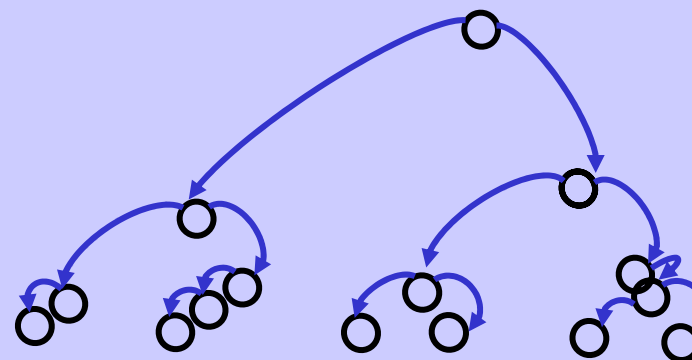
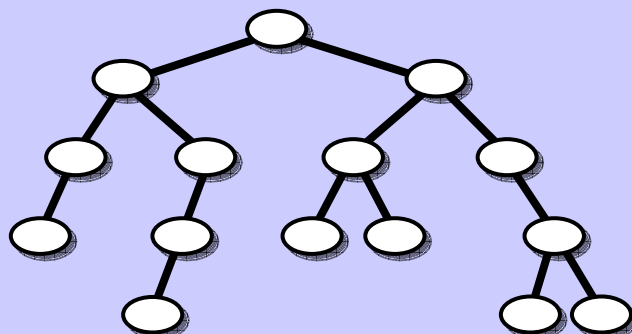


Výpočet optimálního BVS

Korespondence stromů



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



Algoritmus a program není totéž

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

DSA

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

$A_n: (a_1, a_2, \dots, a_n)$

$B_m: (b_1, b_2, \dots, b_m)$

$C_k: (c_1, c_2, \dots, c_k)$

.....
 $C_k = \text{LCS}(A_n, B_m)$

	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:

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

	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

$$(a_n \neq b_m) \ \& \ (c_k \neq a_n) \implies (C_k = \text{LCS}(A_{n-1}, B_m))$$

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

	1	2	3	4	5	6	7	8
A_6 :	C	B	E	A	D	D	E	
B_6 :	D	E	C	D	B	D		
C_3 :	E	D	D					

$$(a_n \neq b_m) \ \& \ (c_k \neq b_m) \implies (C_k = \text{LCS}(A_n, B_{m-1}))$$

	1	2	3	4	5	6	7	8
A_5 :	C	B	E	A	D			
B_5 :	D	E	C	D	B			
C_2 :	E	D						

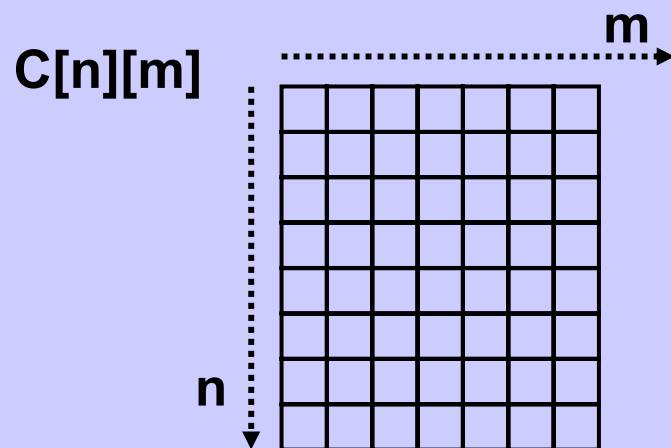
	1	2	3	4	5	6	7	8
A_5 :	C	B	E	A	D			
B_4 :	D	E	C	D	B			
C_2 :	E	D						

Nejdelší společná podposloupnost

Rekurzivní funkce – délka NSP

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

Strategie dynamického programování



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

Nejdelší společná podposloupnost

Konstrukce pole pro NSP

```
void findLCS() {  
    int a, b;  
    for (a=1; a<=n; a++)  
        for (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

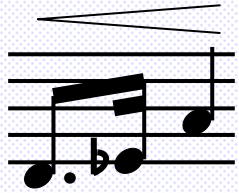
Pole
NSP
pro
“CBEADDEA”
a
“DECDBDA”

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
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

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);  
}
```



DSA

Různé algoritmy mají různou složitost

Algoritmus a program není totéž