Proiectarea algoritmilor: Programare dinamică

Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

PA 2013/2014

1. Prezentarea generală a paradigmei

2. Studii de caz
   - Problema rucsacului II (varianta discretă)
   - Distanța între șiruri

Plan

1. Prezentarea generală a paradigmei

2. Studii de caz
   - Problema rucsacului II (varianta discretă)
   - Distanța între șiruri

Clasa de probleme

Clasa de probleme la care se aplică include probleme de optim.

Modelul matematic

1. Definirea noțiunii de **stare**, care este de fapt o generalizare a problemei inițiale și asocierea funcției obiectiv pentru stare. Problemei trebuie sa devină un caz particular (o stare).
2. Definirea unei relații de tranziție între stări. O relație $s \to s'$, unde s și s' sunt stări, va fi numită **decizie**. O *politică* este o secvență de decizii consecutive, adică o secvență de forma $s_0 \to s_1 \to \cdots \to s_n$.
3. Unei stări s îi asociem o valoare z și definim $f(z)$ astfel încât, dacă starea s corespunde problemei inițiale, atunci

$$f(z) = \text{optim } R(x_0, \ldots, x_{m-1})$$

Modelul matematic

4. Aplicarea **Principiului de Optim** pentru obține o relație de recurență. Principiul de optim (PO) afirmă că o subpolitică a unei politici optimale este la rândul ei optimală. Deoarece este posibil ca PO să nu aibă loc, rezultă că trebuie verificată validitatea relației de recurență.

5. Principiul de optim conduce la obținerea unei ecuații funcționale de forma:

$$f(z) = \operatorname*{optim}_{y} \left[H(z, y, f(T(z, y))) \right] \tag{1}$$

   unde:

   - s și s' sunt două stări astfel încât una se obține din cealaltă aplicând decizia d,
   - z este valoarea asociată stării s,
   - $T(z, y)$ calculează valoarea stării s', iar
   - H exprimă algoritmul de calcul al valorii $f(z)$ dat de decizia d.

Modelul matematic

6. Relația de mai sus poate fi interpretată astfel: dintre toate deciziile care se pot lua în starea s (sau care conduc la starea s), se alege una care dă valoarea optimă în condițiile în care politica aplicată în continuare (sau până atunci) este este și ea optimă.

7. Relația 1 poate fi dovedită utilizând inducția și reducerea la absurd. Deoarece este posibil ca principiul de optim să nu aibă loc pentru anumite formulări, este necesară verificarea sa pentru problema supusă rezolvării.

8. Rezolvarea ecuațiilor recurente 1 conduce la determinarea unui șir de decizii ce în final constituie politica optimă prin care se determină valoarea funcției obiectiv.

Modelul matematic

9. **Calculul recurenței** rezolvând problemele de la mic la mare și memorând valorile obținute într-un tablou. Nu se recomandă scrierea unui program recursiv care să calculeze valorile optime. Dacă în secvența de decizii luate, o anumită problemă apare de mai multe ori, ea va fi calculată de algoritmul recursiv de câte ori apare.
   Exemplu cu numerele Fibonacci: pe tabla
10. Extragerea soluției optime din tablou utilizând proprietatea de substructură optimă a soluției, care afirmă că *soluția optimă a problemei include soluțiile optime ale subproblemelor sale*. De remarcat că proprietatea de substructură optimă este echivalentă cu principiul de optim.

Plan

1. Prezentarea generală a paradigmei

2. Studii de caz
   - Problema rucsacului II (varianta discretă)
   - Distanța între șiruri

Plan

Problema

Se consideră n obiecte $1, \ldots, n$ *de dimensiuni (greutăți)* $w_1, \ldots, w_n \in \mathbb{Z}_+$*, respectiv, și un rucsac de capacitate* $M \in \mathbb{Z}_+$*. Un obiect i sau este introdus în totalitate în rucsac,* $x_i = 1$*, sau nu este introdus de loc,* $x_i = 0$*, astfel că o umplere a rucsacului constă dintr-o secvență* $x_1, \ldots, x_n$ *cu* $x_i \in \{0, 1\}$ *și* $\sum_i x_i \cdot w_i \leq M$*. Ca și în cazul continuu, introducerea obiectului i în rucsac aduce profitul* $p_i \in \mathbb{Z}$ *iar profitul total este* $\sum_{i=1}^{n} x_i p_i$*. Problema constă în a determina o alegere* $(x_1, \ldots, x_n)$ *care să aducă un profit maxim.*

Deci, singura deosebire față de varianta continuă studiată la metoda greedy constă în condiția $x_i \in \{0, 1\}$ în loc de $x_i \in [0, 1]$.

Formularea matematică

- funcția obiectiv:

$$\max \sum_{i=1}^{n} x_i \cdot p_i$$

- restricții:

$$\sum_{i=1}^{n} x_i \cdot w_i \leq M$$
$$x_i \in \{0,1\}, i = 1, \ldots, n$$
$$w_i \in \mathbb{Z}_+, p_i \in \mathbb{Z}, i = 1, \ldots, n$$
$$M \in \mathbb{Z}$$

Definirea noțiunii de stare

$\textsc{Rucsac}(j, X)$ reprezintă următoarea problemă, care este o generalizare a celei inițiale:

- funcția obiectiv:

$$\max \sum_{i=1}^{j} x_i \cdot p_i$$

- restricții:

$$\begin{array}{l} \sum_{i=1}^{j} x_i \cdot w_i \leq X \\ x_i \in \{0, 1\}, i = 1, \ldots, j \\ w_i \in \mathbb{Z}_+, p_i \in \mathbb{Z}, i = 1, \ldots, j \\ X \in \mathbb{Z} \end{array}$$

Cu $f_j(X)$ notăm valoarea optimă pentru instanța $\textsc{Rucsac}(j, X)$. Dacă $j = 0$ și $X \geq 0$, atunci $f_j(X) = 0$.

Definirea noțiunilor de decizie și relația de tranziție

Presupunem $j > 0$. Considerăm decizia optimă prin care starea $\textsc{Rucsac}(j, X)$ este transformată în $\textsc{Rucsac}(j-1, ?)$.

Notăm cu $(x_1, \ldots, x_j)$ alegerea care dă valoarea optimă $f_j(X)$.

Aplicarea principiului de optim

Dacă $x_j = 0$ (obiectul j nu este pus în rucsac) atunci, conform principiului de optim, $f_j(X)$ este valoarea optimă pentru starea $\textsc{Rucsac}(X, j-1)$ și de aici $f_j(X) = f_{j-1}(X)$.

Dacă $x_j = 1$ (obiectul j este pus în rucsac) atunci, din nou conform principiului de optim, $f_j(X)$ este valoarea optimă pentru starea $\textsc{Rucsac}(j-1, X - w_j)$ la care se adaugă profitul p_i și de aici $f_j(X) = f_{j-1}(X - w_j) + p_j$.

Obținerea recurenței

Combinând relațiile de mai sus obținem:

$$f_j(X) = \begin{cases} -\infty & \text{, dacă } X < 0 \\ 0 & \text{, dacă } j = 0 \text{ şi } X \geq 0 \\ \max\{f_{j-1}(X), f_{j-1}(X - w_j) + p_j\} & \text{, dacă } j > 0 \text{ şi } X \geq 0 \end{cases} \quad (2)$$

Am considerat $f_j(X) = -\infty$ dacă $X < 0$.

Rezolvarea problemelor de la mic la mare și memorarea rezultatelor în tabel

Fie $M = 10, n = 3$ și greutățile și profiturile date de următorul tabel:

i	1	2	3
w_i	3	5	6
p_i	10	30	20

Valorile optime pentru subprobleme sunt calculate cu ajutorul relațiilor 2 și pot fi memorate într-un tablou bidimensional astfel:

X	0	1	2	3	4	5	6	7	8	9	10
f_0	0	0	$\boxed{0}$	0	0	0	0	0	0	0	0
f_1	0	0	0	10	10	$\boxed{10}$	10	10	10	10	10
f_2	0	0	0	10	10	30	30	30	40	40	$\boxed{40}$
f_3	0	0	0	10	10	30	30	30	40	40	$\boxed{40}$

Tabloul de mai sus este calculat linie cu linie: pentru a calcula valorile de pe o linie sunt consultate numai valorile de pe linia precedentă.

Analiza

Tabloul de mai sus are dimensiunea $n \cdot m$ (au fost ignorate prima linie și prima coloană). Dacă $m = O(2^n)$ rezultă că atât complexitatea spațiu cât și cea timp sunt exponențiale. Privind tabloul de mai sus observăm că există multe valori care se repetă.

Rafinare

În continuare ne punem problema memorării mai compacte a acestui tablou. Construim graficele funcțiilor f_0, f_1, f_2 și f_3 pentru exemplul de mai sus. Avem:

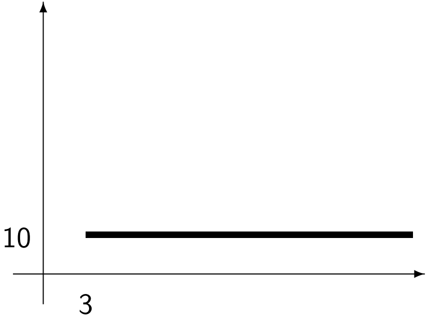
$$f_0(X) = \begin{cases} -\infty & , X < 0 \\ 0 & , X \geq 0 \end{cases}$$

Notăm cu g_0 funcția dată de:

$$g_0(X) = f_0(X - w_1) + p_1 = \begin{cases} -\infty & , X < 3 \\ 10 & , 3 \leq X \end{cases}$$

Rafinare

Graficele funcțiilor f_0 și g_0:



Rafinare

Funcția f_1 se calculează prin:

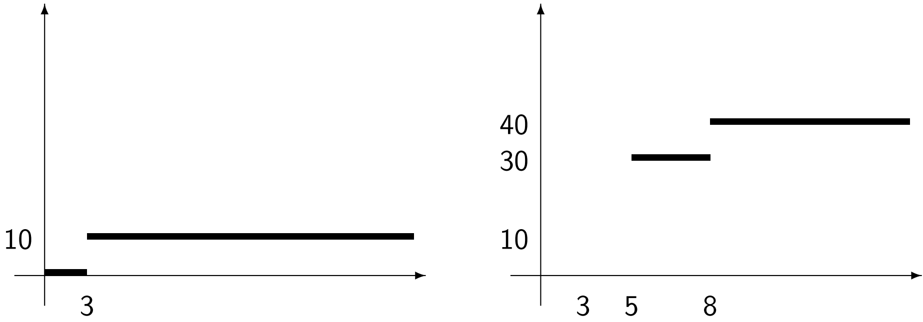
$$f_1(X) = \max\{f_0(X), g_0(X)\} = \begin{cases} -\infty & , X < 0 \\ 0 & , 0 \leq X < 3 \\ 10 & , 3 \leq X \end{cases}$$

Notăm cu g_1 funcția dată prin:

$$g_1(X) = f_1(X - w_2) + p_2 = \begin{cases} -\infty & , X < 5 \\ 30 & , 5 \leq X < 8 \\ 40 & , 8 \leq X \end{cases}$$

Rafinare

Graficele funcțiilor f_1 și g_1:



Rafinare

Funcția f_2 se calculează prin:

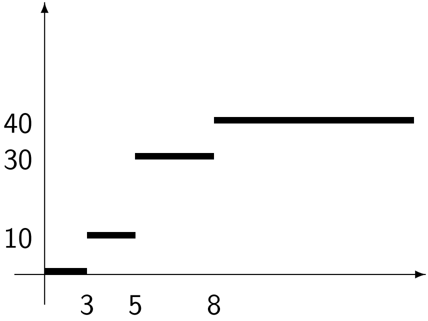
$$f_2(X) = \max\{f_1(X), g_1(X)\} = \begin{cases} -\infty & , X < 0 \\ 0 & , 0 \leq X < 3 \\ 10 & , 3 \leq X < 5 \\ 30 & , 5 \leq X < 8 \\ 40 & , 8 \leq X \end{cases}$$

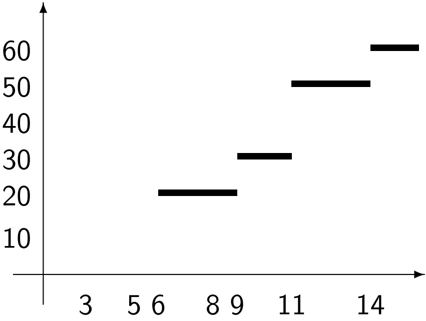
În continuare, notăm cu g_2 funcția dată prin:

$$g_2(X) = f_2(X - w_3) + p_3 = \begin{cases} -\infty & , X < 6 \\ 20 & , 6 \leq X < 9 \\ 30 & , 9 \leq X < 11 \\ 50 & , 11 \leq X < 14 \\ 60 & , 14 \leq X \end{cases}$$

Rafinare

Graficele funcțiilor f_2 și g_2:





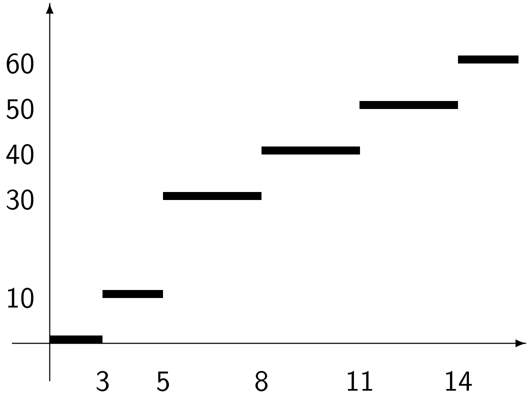
Rafinare

Funcția f_3 se calculează prin:

$$f_3(X) = \max\{f_2(X), g_2(X)\} = \begin{cases} -\infty & , X < 0 \\ 0 & , 0 \leq X < 3 \\ 10 & , 3 \leq X < 5 \\ 30 & , 5 \leq X < 8 \\ 40 & , 8 < X \leq 11 \\ 50 & , 11 \leq X < 14 \\ 60 & , 14 \leq X \end{cases}$$

Rafinare

Graficul funcției f_3:



Rafinare

Se remarcă faptul că funcțiile f_i și g_i sunt funcții în scară. Graficele acestor funcții pot fi reprezentate prin mulțimi finite din puncte din plan. De exemplu, graficul funcției f_2 este reprezentat prin mulțimea $\{(0,0),(3,10),(5,30),(8,40)\}$.

X	0	1	2	3	4	5	6	7	8	9	10
f_0	0	0	0	0	0	0	0	0	0	0	0
f_1	0	0	0	10	10	10	10	10	10	10	10
f_2	0	0	0	10	10	30	30	30	40	40	40
f_3	0	0	0	10	10	30	30	30	40	40	40

O mulțime care reprezintă o funcție în scară conține acele puncte în care funcția face salturi. Graficul funcției g_i se obține din graficul funcției f_i printr-o translație iar graficul funcției f_{i+1} se obține prin interclasarea graficelor funcțiilor f_i și g_i.

Rafinare

În general, fiecare f_i este complet specificat de o mulțime $S_i = \{(X_j, Y_j) \mid j = 0, \ldots, r\}$ unde $Y_j = f_i(X_j)$.

Presupunem $X_1 < \cdots < X_r$.

Analog, funcțiile g_i sunt reprezentate prin mulțimile $T_i = \{(X + w_i, Y + p_i) \mid (X, Y) \in S_i\}$.

Notăm $T_i = \tau(S_i)$ și $S_{i+1} = \mu(S_i, T_i)$. Mulțimea S_{i+1} se obține din S_i și T_i prin interclasare.

Se consideră o variabilă L care ia valoarea 1 dacă graficul lui f_{i+1} coincide cu cel al lui f_i și cu 2 dacă el coincide cu cel al lui g_i. Deoarece $(0, 0)$ aparține graficului rezultat, considerăm $L = 1, j = 1$ și $k = 1$.

Rafinare

Presupunând că la un pas al interclasării se compară $(X_j, Y_j) \in S_i$ cu $(X_k, Y_k) \in T_i$ atunci:

- dacă $L = 1$:
  - dacă $X_j < X_k$ atunci adaugă (X_j, Y_j) în S_{i+1} și se incrementează j;
  - dacă $X_j = X_k$:
    - dacă $Y_j > Y_k$ atunci adaugă (X_j, Y_j) în S_{i+1} și se incrementează j și k;
    - dacă $Y_j < Y_k$ atunci adaugă (X_k, Y_k) în S_{i+1}, $L = 2$ și se incrementează j și k;
  - dacă $X_j > X_k$ atunci, dacă $Y_k > Y_j$ adaugă (X_k, Y_k) în S_{i+1}, $L = 2$ și se incrementează k;
- dacă $L = 2$:
  - dacă $X_j < X_k$ atunci, dacă $Y_j > Y_k$ adaugă (X_j, Y_j) în S_{i+1}, $L = 1$ și se incrementează j;
  - dacă $X_j = X_k$:
    - dacă $Y_j < Y_k$ atunci adaugă (X_k, Y_k) în S_{i+1} și se incrementează j și k;
    - dacă $Y_j > Y_k$ atunci adaugă (X_j, Y_j) în S_{i+1}, $L = 1$ și se incrementează j și k;
  - dacă $X_j > X_k$ atunci adaugă (X_k, Y_k) în S_{i+1} și se incrementează k;

Rafinare

Rămâne de extras soluția optimă din S_n. Considerăm mai întâi cazul din exemplul de mai sus.

- Se caută în $S_n = S_3$ perechea (X_j, Y_j) cu cel mai mare X_j pentru care $X_j \leq M$. Obținem $(X_j, Y_j) = (8, 40)$. Deoarece $(8, 40) \in S_3$ și $(8, 40) \in S_2$ rezultă $f_{optim}(M) = f_{optim}(8) = f_3(8) = f_2(8)$ și deci $x_3 = 0$. Perechea (X_j, Y_j) rămâne neschimbată.
- Pentru că $(X_j, Y_j) = (8, 40)$ este în S_2 și nu este în S_1, rezultă că $f_{optim}(8) = f_1(8 - w_2) + p_2$ și deci $x_2 = 1$. În continuare se ia $(X_j, Y_j) = (X_j - w_2, Y_j - p_2) = (8 - 5, 40 - 30) = (3, 10)$.
- Pentru că $(X_j, Y_j) = (3, 10)$ este în S_1 și nu este în S_0, rezultă că $f_{optim}(3) = f_1(3 - w_1) + p_1$ și deci $x_1 = 1$.

Rafinare

Metoda poate fi descrisă pentru cazul general:

- Inițial se determină perechea $(X_j, Y_j) \in S_n$ cu cel mai mare X_j pentru care $X_j \leq M$. Valoarea Y_j constituie încărcarea optimă a rucsacului, i.e. valoarea funcției obiectiv din problema inițială.
- Pentru $i = n - 1, \ldots, 0$:
    - dacă (X_j, Y_j) este în S_i, atunci $f_{i+1}(X_j) = f_i(X_j) = Y_j$ și se face $x_{i+1} = 0$ (obiectul $i + 1$ nu este ales);
    - dacă (X_j, Y_j) nu este în S_i, atunci $f_{i+1}(X_j) = f_i(X_j - w_{i+1}) + p_{i+1} = Y_j$ și se face $x_{i+1} = 1$ (obiectul $i + 1$ este ales), $X_j = X_j - w_{i+1}$ și $Y_j = Y_j - p_{i+1}$.

Algoritmul rafinat

```
rucsacVD(n, w, p, valOpt, x) {
   S₀ = {(0,0)};
   T₀ = {(w₁,p₁)};
   for (i = 1; i < n; ++i) {
       Sᵢ(X) = μ(Sᵢ₋₁, Tᵢ₋₁)
       Tᵢ = {(X + wᵢ, Y + pᵢ) | (X,Y) ∈ Sᵢ}
   }
   determină (Xⱼ,Yⱼ) cu Xⱼ = max{Xᵢ | (Xᵢ,Yᵢ) ∈ Sₙ, Xᵢ ≤ M}
   for (i = n-1; i ≥ 1; --i)
       if ((Xⱼ,Yⱼ) ∈ Sᵢ) then xᵢ₊₁ = 0
       else {
          xᵢ₊₁ = 1
          Xⱼ = Xⱼ − wᵢ₊₁
          Yⱼ = Yⱼ − pᵢ₊₁
       }
}
```

Analiza algoritmului rafinat

Notăm $m = \sum_{i=0}^{n} |S_i|$. Deoarece $|T_i| = |S_i|$ rezultă că $|S_{i+1}| \leq 2 \cdot |S_i|$ și de aici $\sum_i |S_i| \leq \sum_i 2^i = 2^n - 1$. Calculul lui S_i din S_{i-1} necesită timpul $\Theta(|S_{i-1}|)$ și de aici calculul lui S_n necesită timpul $\sum_i \Theta(|S_i|) = O(2^n)$. Deoarece profiturile p_i sunt numere întregi, pentru orice $(X, Y) \in S_i$, Y este întreg și $Y \leq \sum_{j \leq i} p_j$. Analog, pentru că dimensiunile w_i sunt numere întregi, pentru $(X, Y) \in S_i$, X este întreg și $X \leq \sum_{j \leq i} w_j$. Deoarece perechile (X, Y) cu $X > M$ nu interesează, ele pot să nu fie incluse în mulțimile S_i.

Analiza algoritmului rafinat

De aici rezultă că numărul maxim de perechi (X, Y) distincte din S_i satisface relațiile:

$$|S_i| \leq 1 + \textstyle\sum_{j=1}^{i} w_j$$
$$|S_i| \leq M$$

care implică

$$|S_i| \leq 1 + \min\left\{\sum_{j=1}^{i} w_j, M\right\}$$

Relația de mai sus permite o estimare mai precisă a spațiului necesar pentru memorarea mulțimilor S_i în cazul unor probleme concrete. În ceea ce privește timpul, făcând calculele rezultă că algoritmul are complexitatea timp $O(\min(2^n, n\sum_{i=1}^{n} p_i, nM))$.

Plan

1. Prezentarea generală a paradigmei

2. Studii de caz
   - Problema rucsacului II (varianta discretă)
   - Distanța între șiruri

Problema

Se consideră două șiruri $\alpha = a_1 \cdots a_n$ și $\beta = b_1 \cdots b_n$ formate cu litere dintr-un alfabet A. Asupra șirului α se pot face următoarele operații:

- Ștergere: *$S(i)$ – șterge litera de pe poziția i;*
- Inserare: *$I(i, c)$ – inserează litera c pe poziția i;*
- Modificare: *$M(i, c)$ – înlocuiește litera de pe poziția i cu c.*

Problema constă în determinarea unei secvențe de operații de lungime minimă care transformă pe α în β.

Exemplu: Fie $\alpha = carnet, \beta = paleta$. O secvență de transformări este $carnet \mapsto parnet \mapsto palnet \mapsto palet \mapsto paleta$. Transformările efectuate sunt $M(1, p), M(3, l), S(4)$ și $I(6, a)$. Există o altă secvență mai scurtă?

$\boxed{\text{sfex}}$

Analiza domeniului problemei

Lemă

Fie $s = (\ldots T(i, \_), T'(j, \_) \ldots)$ *o secvență optimă (de lungime minimă) care transformă pe* α *în* β*. Atunci există* k, ℓ *astfel încât secvența* $s' = (\ldots T'(k, \_), T(\ell, \_) \ldots)$*, obținută din* s *prin interschimbarea celor două operații* T *și* T' *și în rest rămânând neschimbată, este de asemenea o secvență optimă care transformă pe* α *în* β*.*

Corolar

Există o secvență optimă care dacă efectuează transformările:

$$\alpha = \alpha_0 \mapsto \alpha_1 \mapsto \cdots \mapsto \alpha_t = \beta$$

atunci pentru orice i, $|\alpha_i| \leq |\beta|$*, unde prin* $|\_|$ *am notat lungimea șirului* $\_$*.*

Analiza domeniului problemei

Lemă

Are loc:

(i) $d(\alpha, \alpha) = 0$;

(ii) $d(\alpha, \beta) = d(\beta, \alpha)$;

(iii) $d(\alpha, \gamma) \leq d(\alpha, \beta) + d(\beta, \gamma)$.

Observație: Lema de mai sus arată că $d(\alpha, \beta)$ este o metrică. De aici și numele problemei. $\boxed{\text{sfobs}}$

Noțiunea de stare

$DS(\alpha_i, \beta_j)$ corespunzătoare transformării subșirului $\alpha_i = a_1 \dots a_i$ în $\beta_j = b_1 \dots b_j$ și prin $d[i, j]$ valoarea optimă $d(\alpha_i, \beta_j)$.

Dacă $i = 0$, α_0 este șirul vid și β_j se obține prin j inserări: deci $d[0, j] = j$.

Dacă $j = 0$, atunci β_j se obține prin i ștergeri și avem $d[i, 0] = i$.

În continuare presupunem $i, j > 0$.

Decziile și Principiul de optim

Considerăm decizia optimă prin care starea $DS(a_1 \dots a_i, b_1 \dots b_j)$ este transformată într-o stare $DS(a_1 \dots a_{i'}, b_1 \dots b_{j'})$ cu ($i' < i$ și $j' \leq j$) sau ($i' \leq i$ și $j' < j$).

Distingem următoarele situații:

1. Dacă $a_i = b_j$ atunci $i' = i - 1, j' = j - 1$ și, aplicând principiul de optim, obținem $d[i, j] = d[i - 1, j - 1]$.
2. $DS(a_1, \dots, a_i, b_1, \dots, b_j)$ se obține prin ștergere. Rezultă $i' = i - 1, j' = j$ și, aplicând principiul de optim, obținem $d[i, j] = d[i - 1, j] + 1$.
3. $DS(a_1, \dots, a_i, b_1, \dots, b_j)$ se obține prin inserare. Avem $i' = i, j' = j - 1$ și, aplicând principiul de optim, obținem $d[i, j] = d[i, j - 1] + 1$. Din corolarul lemei precedente rezultă că această operație poate fi realizată numai dacă $i < j$.
4. $DS(a_1, \dots, a_i, b_1, \dots, b_j)$ se obține prin modificare. Avem $i' = i - 1, j' = j - 1$ și, aplicând principiul de optim, obținem $d[i, j] = d[i - 1, j - 1] + 1$.

Relația de recurență

Deoarece $d[i,j]$ trebuie să fie minimă, rezultă:

$$d[i,j] = \min\{d[i-1,j]+1, d[i-1,j-1]+\delta, d[i,j-1]+1\}$$

unde

$$\delta = \begin{cases} 0 & \text{, dacă } a_i = b_j \\ 1 & \text{, dacă } a_i \neq b_j \end{cases}$$

Determinarea secvenței optime

Notăm cu L lista care înregistrează transformările soluției optime. Se procedează în modul următor:

- inițial se consideră $i = n$; $j = n$; și $L = (\ )$ (lista vidă);
- următorul proces se repetă până când i și j devin 0:
  - dacă $d[i, j] = d[i-1, j-1]$ atunci L rămâne neschimbată și se face `i = i-1; j = j-1;`
  - altfel, dacă $d[i, j] = d[i-1, j-1] + 1$ atunci se face $L = (M(i, b_j), L)$ și `i = i-1; j = j-1;`
  - altfel, dacă $d[i, j] = d[i-1, j] + 1$ atunci se face $L = (S(i), L)$ și `i  = i-1;`
  - altfel, dacă $d[i, j] = d[i, j-1] + 1$ atunci se face $L = (I(i, b_j), L)$ și `j = j-1`.

Algoritmul

```
distSir(a, b, n) {
   for (j = 1; j ≤ n; ++j) d[0,j] = j;
   for (i = 1; i ≤ n; ++i) d[i,0] = i;
   for (i = 1; i ≤ n; ++i)
       for (j = 1; j ≤ n; ++j) {
           δ = (a[i] == b[j])?  0 :  1;
           d[i,j] = min{d[i − 1,j] + 1, d[i − 1,j − 1] + δ, d[i,j − 1] + 1}
       }
   L = listaVida();
   i = n; j = n;
   repeat
      if (d[i,j] == d[i-1,j-1]) {
         i = i-1; j = j-1;
      } else if (d[i,j] == d[i-1,j-1]+1) {
           L.pushFront('M', i, b[j]));
           i = i-1; j = j-1;
      } else if (d[i,j] == d[i-1,j]+1) {
           L.pushFront('S', i));
           i = i-1
      } else {
           L.pushFront('I', i, b[j]);
           j = j-1;
      }
   until ((i = 0) && (j = 0))
}
```

Analiza

Exemplu: dist-sir-ex.pptx

Teoremă

Determinarea distanței optime și a secvenței optime care transformă un șir α într-un șir β, $|\alpha| = |\beta| = n$, se poate face în timpul $O(n^2)$.

Variații

- alte operații:
  transpoziția: schimbă ordinea a două caractere adiacente
- distanța Levenshtein (de editare)
  sunt admise numai inserari, stergeri si inlocuiri
  toate operațiile au costul 1
- distanța Hamming
  sunt admise numai înlocuirile
  costul operației este 1
  este finita ori de cate ori $|a| = |b|$
- distanta episodica (episode distance)
  sunt admise numai inserări
  costul operației este 1
  distanța este sau $|b| - |a|$ sau ∞

Variații

- distanța data de cea mai lungă subsecvență
  sunt admise numai inserări și ștergeri
  toate operațiile au costul 1
  Exemplu: $\alpha =$ amxbtycsnma și $\beta =$ bancxstymcxn
  $\alpha =$ amxbtycsnma $\to$ bamxbtycsnma $\to$ baxbtycsnma $\to$
  banxbtycsnma $\to$ bancxbtycsnma $\to$ bancxtycsnma $\to$
  bancxstycsnma $\to$bancxstymcsnma $\to$ bancxstymcnma $\to$
  bancxstymcxnma $\to$ bancxstymcxna $\to$ bancxstymcxn $= \beta$
  (a,x,t,y,c,n) este subsecventa comună
  este cea mai lungă?

Variații

- matching aproximativ peste șiruri (aproximate string matching) problema: dat un text s de lungime n, un patern p de lungime m, o distanța $d()$ între șiruri și un număr k, să se determine pozițiile j din textul s astfel încât să existe i cu $d(p, s[i..j]) \leq k$
  - distanța Levenshtein: string matching with k differences
  - distanța Hamming: string matching with k missmatches
  - distanța episodica: episode matching (modeleaza cazul cand se cauta o secventa de evenimente intr-o perioada scurta de timp)
  - cea mai lungă subsecvență comună: exact ce spune numele

Variații

procesul de cautare pentru matching aproximativ:

- $\alpha = p$, $\beta = s$
  trebuie să modificăm algoritmul a.î. orice poziție j din text este startul potențial al unei potriviri; asta se realizează prin setarea $d[0, j] = 0$
- calculul matricei se face pe coloane
- inițial: $d[i, 0] = i$ pentru $i = 0, \ldots, m$
- se procesează textul caracter cu caracter
- presupunem că la pasul curent se procesează s_j
- coloana j este actualizată:
  $d[i, j] = (p_i == s_j)\ ?\ d[i-1, j-1] :$
  $1 + min(d[i-1, j], d[i, j-1], d[i-1, j-1])$
- pozitiile j pentru care $d[m, j] \leq k$ sunt raportate
- de remarcat că numai ultimele două coloane sunt necesare