4.1. Úvod do dynamického programovania (Dynamic programming)

Princíp rozdeľuj-a-panuj(divide-and-conquer), ktorý sa používa pri mnohých problémoch, pracuje nasledovne: veľký problém rozdelíme na viaceré samostatné menšie podproblémy, ktoré vyriešime a výsledky použijeme (skombinujeme) na vytvorenie riešenia celého veľkého problému. V dynamickom programovaní tento princíp dovedieme do extrému: keď presne nevieme, ktoré podproblémy treba vyriešiť, jednoducho ich vyriešime všetky, výsledky si zapamätáme a opäť využijeme na doriešenie väčších problémov.

4.1.1. Kedy použiť dynamické programovanie?

Pri aplikovaní dynamického programovanie musíme počítať s niektorými úskaliami. Po prvé, nie vždy je možné skombinovať riešenia menších problémov na vytvorenie riešenia väčšieho problému a po druhé, počet malých problémov, ktoré potrebujeme vyriešiť môže byť príliš veľký. Nedá sa precízne vyčleniť, ktoré problémy sa dajú vhodne takto riešiť, ale sú určité ukazovatele, ktoré nám pomáhajú:

- Optimálna podštruktúra
- Prelínajúce sa podproblémy

Prvý krok pri riešení optimalizačných problémov (hľadáme najlepšie riešenie) je charakterizovať to hľadané najlepšie riešenie. Hovoríme, že problém v sebe obsahuje ***optimálnu podštruktúru*** práve vtedy keď, každé najlepšie riešenie (môže ich byť predsa aj viac) v sebe obsahuje optimálne riešenia podproblémov (napríklad najkratšia cesta v grafe obsahuje len najkratšie podcesty).

Ďalším dôležitým ukazovateľom, že by sme mohli použiť dynamické programovanie je ***prelínanie podproblémov***. Teda, že množstvo rôznych podproblémov, ktoré treba vyriešiť je malé. Keď rekurzívne riešenie stále prehľadáva rovnaké a rovnaké podproblémy, tak treba začať uvažovať nad využitím dynamického programovania. Naproti tomu, problém na ktorý je vhodná technika rozdeľuj-a-panuj generuje v každom kroku rekurzie celkom nové podproblémy a teda by sme si ich len zbytočne pamätali. Riešenie založené na dynamickom programovaní zvykne využívať výsledky už vyriešených podproblémov, aby ich, neskôr, nemuselo počítať viac razy.

4.1.2. Technika pamätania(memoization)

Typické dynamické programovanie postupuje v hierarchii problémov od najmenších po najväčšie. Variácia dynamického programovania, ktorá zachováva prirodzenú orientáciu od väčších problémov k menším sa nazýva technika pamätania. Myšlienka spočíva v zaznamenávaní výsledkov, ktoré už raz rekurzívny (neefektívny) program vypočítal a vždy, keď by ich mal počítať znovu, si ich už len prečíta z tabuľky, kam si ich zapamätal.

4.1.3. Konštrukcia riešenia

Algoritmus pracujúci na princípe dynamického programovania sa dá myšlienkovo rozdeliť na štyri časti, z ktorých sa štvrtá občas zanedbáva(nie vždy nás zaujíma konkrétne riešenie, stačí nám jeho hodnota) :

1. Charakterizovanie optimálneho riešenia
2. Rekurzívna definícia optimálneho riešenia
3. Výpočet riešenia postupne smerom od menších podproblémov po väčšie
4. Konštrukcia optimálneho riešenia z vypočítanej informácie

4.1.4 Najdlhšia (vybraná) spoločná podpostupnosť(Longest common subsequence)
Teória dynamického programovania je jednoduchá. Na zvládnutie tohto postupu treba preriešiť mnoho modelových úloh. My sa teraz budeme zaoberať nájdením najdlhšej spoločnej podpostupnosti dvoch postupností.

Podpostupnosť postupnosti *X* je *X* s niektorým (aj žiadnymi) prvkami vynechanými. Predstavme si teraz dve postupnosti *X* a *Y*. *Z* je spoločná podpostupnosť *X* a *Y* práve vtedy, keď je podpostupnosťou každej z nich. Keď neexistuje spoločná podpostupnosť s viacerými prvkami, hovoríme že *Z* je najdlhšia spoločná podpostupnosť (LCS). Dané *X* a *Y*, zaujíma nás nájsť najdlhšiu spoločnú podpostupnosť.

Aplikujeme dynamické programovanie: rekurzívne budeme počítať LCS postupností *X* a *Y* tak, že budeme postupne uvažovať predlžujúce sa LCS dvojíc prefixov *X* a *Y*. Nech X_n je postupnosť $< x_1, x_2, ..., x_n >$ a Y_m je postupnosť $< y_1, y_2, ..., y_m >$. Naviac, nech $Z_k =< z_1, z_2, ..., z_k >$ je ľubovoľná LCS postupností X_n a Y_m. Potom zrejme platia nasledujúce tvrdenia:

- Ak $x_n = y_m$, potom $z_k = x_n = y_m$ a Z_{k-1} je LCS postupností X_{n-1} a Y_{m-1}
- Ak $x_n \neq y_m$, potom ak $z_k \neq x_n$, Z_k musí byť LCS postupností X_{n-1} a Y_m
- Ak $x_n \neq y_m$, potom ak $z_k \neq y_m$, Z_k musí byť LCS postupností X_n a Y_{m-1}

Všimnime si, že na vyriešenie problému pre (X_n, Y_m) potrebujeme vedieť výsledky pre (X_{n-1}, Y_m), (X_n, Y_{m-1}) a (X_{n-1}, Y_{m-1}). Pri rekurzívnom riešení, aby sme nemuseli v rôznych vetvách riešiť rovnaké podproblémy viackrát, použijeme techniku pamätania. Priamočiare dynamické programovanie funguje ale iným spôsobom a to takým, že vždy keď potrebujeme mať niečo už vypočítané, tak to už máme je vypočítané – postupujeme od menších problémov k väčším.

Kostra nášho „rýdzeho“ riešenia vyzerá potom nasledovne:

```
for i := 0 to n do
  for j := 0 to m do
    lcs[i,j] := "dĺžka LCS postupností Xi a Yj"
```

Táto procedúra vypĺňa tabuľku `lcs`, kde `lcs[i,j]` obsahuje dĺžku LCS postupností X_i a Y_j. Uvedomme si že, v čase keď počítame `lcs[i,j]` boli všetky menšie podproblémy už vyriešené.

Nech teda máme vypočítané dĺžky najdlhších spoločných podpostupností pre všetky možné začiatky postupností *X* a *Y*, potom vieme výslednú spoločnú podpostupnosť už jednoducho skonštruovať.

Pozrime sa teraz na poradie vypĺňania jednotlivých políčok tabuľky. V i-tej iterácií i-cyklu vyplníme i-ty riadok tabuľky s tým, že potrebujeme vedieť len výsledky predchádzajúceho riadku. Tuto môžeme ušetriť nejakú pamäť. V prípade, že nás nezaujíma výsledná podpostupnosť, ale stačí nám len jej dĺžka, je nám celá tabuľka `lcs` zbytočná, môžeme sa obmedziť len na predchádzajúci a ďalší riadok.

Priamočiaro, uvedený algoritmus má časovú zložitosť $O(NM)$, pamäťová zložitosť závisí na požiadavkách, ktoré kladieme na výsledok.

4.2. Dynamické programovanie II.

Dynamické programovanie sa hojne používa na riešenie rozmanitých problémov. Schopnosť vidieť problém v reči dynamického programovania „nespadne z neba“ ale vyžaduje bezprostredné skúsenosti s množstvom takýchto úloh. Ponúkame prehľad 8 modelových úloh spolu s riešeniami. Odporúčaný postup pri riešení je prečítať si zadanie úlohy a snažiť sa ju vyriešiť bez nahliadnutia do riešenia.

4.2.1. Modelové úlohy

Najdlhšia vybraná rastúca podpostupnosť

Daná je konečná postupnosť celých čísel dĺžky *N*, $N \geq 1$. Prvky tejto postupnosti označíme po rade $x_1, x_2, ..., x_N$. Pod podpostupnosťou dĺžky *k* vybranou z danej postupnosti budeme rozumieť ľubovoľnú konečnú postupnosť tvaru $x_{i_1}, x_{i_2}, ..., x_{i_k}$, kde $1 \leq i_1 < i_2 < ... < i_k \leq N$ (tzv. zo zadanej postupnosť je vybraných ľubovoľných nie nutne susedných *k* čísel, pričom je zachované ich poradie.

Navrhnite algoritmus, ktorý určí dĺžku najdlhšej rastúcej podpostupnosti vybranej zo zadanej postupnosti. To znamená, určí maximálne *k* také, že $x_{i_1} < x_{i_2} < ... < x_{i_k}$, pre nejaké indexy $1 \leq i_1 < i_2 < ... < i_k \leq N$.

Najdlhšia súvislá rastúca podpostupnosť

Daná je konečná postupnosť celých čísel dĺžky *N*, $N \geq 1$. Prvky tejto postupnosti označíme po rade $x_1, x_2, ..., x_N$. Pod súvislou podpostupnosťou dĺžky *k* danej postupnosti budeme rozumieť ľubovoľnú konečnú postupnosť tvaru $x_i, x_{i+1}, ..., x_{i+k-1}$, kde $1 \leq i < i + k - 1 \leq N$ (tzv. zo zadanej postupnosť je vybraných ľubovoľných susedných *k* čísel).

Navrhnite algoritmus, ktorý určí dĺžku najdlhšej súvislej rastúcej podpostupnosti zadanej postupnosti. To znamená, určí maximálne *k* také, že $x_i < x_{i+1} < ... < x_{i+k-1}$, pre index $1 \leq i < i + k - 1 \leq N$.

Problém batohu (Knapsack Problem)

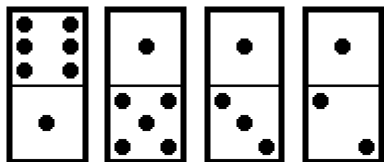
Zlodej, ktorý vylomil zámok na trezore, zistil, že je naplnení *N* vecami rôznych veľkostí a rôznej ceny. Na lup má ale len batoh kapacity *M*, problém batohu je práve nájsť takú kombináciu vecí z trezoru, aby sa mu zmestila do batoha a bola čo možno najcennejšia. Dané sú veľkosti objektov $S_1, S_2, ..., S_N$ a ich ceny $C_1, C_2, ..., C_N$, treba nájsť taký výber objektov $i_1, i_2, ..., i_k$, aby $S_{i_1} + S_{i_2} + ... + S_{i_k} \leq M$ a ich cena $C_{i_1} + C_{i_2} + ... + C_{i_k}$ bola najväčšia možná.

Dominá (Dominoes – CEOI 1997, Nowy Sącz, Poland)

Uvažujme *N* kociek domina naukladaných do radu. Každá kocka má dve čísla A_i a B_i. Spočítajme rozdiel súčtov čísel v hornom riadku a dolnom riadku:

$$\Delta = \left| \sum_{i=1}^{N} A_i - \sum_{i=1}^{N} B_i \right|$$

Jedno otočenie je otočenie jednej kocky domina opačne – výmena jej čísel. Hľadáme najmenší počet otočení taký, aby rozdiel Δ bol najmenší možný.



Pre dominá na obrázku nám postačuje jedno otočenie (otočenie poslednej kocky 1-2 naopak, potom bude súčet v hornom riadku 10, v dolnom riadku 10, takže menej sa určite už nedá).

Plánovanie prednášok(Scheduling Lectures, ACM 1998, EastCentral NorthAmerica)

Na univerzite treba prebrať *N* tém, každá téma vyžaduje $t_1, t_2, ..., t_N$ minút na vysvetlenie. Témy musia byť, kvôli nadväznosti, odučené v poradí 1,2,…,*N*, pričom žiadnu tému nie je možné rozdeliť medzi prednáškami (celú ju treba odučiť v rámci jednej prednášky). Jedna prednáška má dĺžku *L* minút. Niekedy je vhodné mať voľný čas na konci prednášky, ak prednášajúci skončí najviac o 10 minút skôr, študenti sú radi, že môžu odísť skôr, avšak, ak sa skončí skôr, budú sa cítiť, že zo školy nedostávajú dostatok vedomostí. Miera nespokojnosti *DI*(dissatisfaction index) študentov je nasledovná:

- $DI = 0$, ak $t = 0$
- $DI = -C$, ak $1 \le t \le 10$
- inak $DI = (t-10)^2$

kde *C* je kladné celé číslo, *t* je množstvo voľného času na konci prednášky. Celková miera nespokojnosti je súčet mier nespokojností jednotlivých prednášok. Úlohou je rozdeliť témy do prednášok tak, aby sa využilo čo možno najmenej prednášok, a ak je takých rozdelení viac, lepšie je také, ktoré má najmenšiu mieru nespokojnosti študentov. Treba nájsť najlepšie rozdelenie.

Zber jabĺk(Apples, USACO, Fall 2002)

Farmár sa dohodol so svojou kravou, že mu zatrasie jabloňou, a on bude chytať padajúce jablká. Vieme, že, keď krava zatrasie stromom, kam (súradnice [*x,y*]) budú jednotlivé jablká padať a v akom časovom okamihu dopadnú. Farmár chytí jablko vtedy, keď sa bude nachádzať na súradniciach dopadu v momente dopadu. Na začiatok je Farmár na súradniciach [0,0], vieme jeho rýchlosť úlohou je zistiť koľko najviac jabĺk je schopný chytiť.

Palindromické cesty(Palindromic Tax Paths, USACO, Fall 2002)

Daná je mriežka *NxN* pričom v každom mrežovom bode sa nachádza nejaký symbol. Uvažujme mravčeka pohybujúceho sa po mriežke. Z mrežového bodu sa vie presunúť do susedných bodov (aj po diagonále). Začínajúc v ľubovoľnom bode bude chodiť sem a tam, pričom prejde presne *L* bodov. Úlohou je zistiť koľko rôznych palindromických ciest danej dĺžky môže uskutočniť. Zápis cesty nazveme postupnosť symbolov v mrežových bodov v poradí v akom nimi prešiel. Ceste je palindromická vtedy, ak je zápis danej cesty palindrómom, teda ak znie rovnako, keď ho prečítame spredu ako keď ho prečítame zozadu.

Cesta okolo Kanady(Canada Tour, IOI 1993, Mendoza, Argentina)

Vyhrali ste súťaž organizovanú Kanadskými aerolíniami, dostali ste voľný lístok na cestovanie okolo Kanady, začínajúc v najzápadnejšom meste obsluhovanom aerolíniami, cestovaním len zo západu na východ do najvýchodnejšieho mesta a potom zase naspäť. Žiadne mesto sa nesmie navštíviť viac razy, okrem počiatočného, ktoré musí byť navštívené práve dva razy. Nie je povolené používať ani iné prepravné prostriedky ako Kanadské aerolínie. Úlohou je, daných *N* miest, ich poradie od západu na východ a dvojice miest spojené linkou, určiť najväčší počet miest v Kanade, ktoré sa dajú jedným takýmto okružným letom navštíviť.

4.2.2. Riešenia úloh

Najdlhšia vybraná rastúca podpostupnosť

Hľadáme dĺžku najdlhšej vybranej rastúcej podpostupnosti. Z axióm logiky vyplýva, že hľadaná postupnosť musí končiť v nejakom prvku pôvodnej postupnosti. Označme si $D(x_i)$ dĺžku najdlhšej vybranej rastúcej podpostupnosti takej, že končí v prvku x_i pôvodnej postupnosti. Triviálne, keď vieme čísla $D(x_1), D(x_2), ..., D(x_N)$, vieme z nich vybrať najväčšie, čo bude naša hľadaná dĺžka.

Ako ale určíme čísla $D(x_1), D(x_2), ..., D(x_N)$? Postupne. Predpokladajme, že už máme vypočítané čísla $D(x_1), D(x_2), ..., D(x_{i-1})$ pre $i<N$, zaujíma nás vypočítať číslo $D(x_i)$. Všimnime si teraz štruktúru najdlhšej vybranej podpostupnosti končiacej v prvku x_i keďže hľadáme rastúcu vybranú podpostupnosť, platí: $D(x_i) = \max(1, \{D(x_k) \mid x_k < x_i \wedge k < i\})$, čo nám už postačuje na nájdenie našej dĺžky v čase $O(N^2)$.

Poznamenajme, že existuje rýchlejší algoritmus pracujúci v čase $O(N \log N)$, ktorý je jednoduchý, ale natoľko zaujímavý, že ho ponecháme na bádanie čitateľom.

Najdlhšia súvislá rastúca podpostupnosť

Hľadáme dĺžku najdlhšej súvislej rastúcej podpostupnosti. Z axióm logiky vyplýva, že hľadaná postupnosť musí končiť v nejakom prvku pôvodnej postupnosti. Označme si $D(x_i)$ dĺžku najdlhšej súvislej rastúcej podpostupnosti takej, že končí v prvku x_i pôvodnej postupnosti. Triviálne, keď vieme čísla $D(x_1), D(x_2), ..., D(x_N)$, vieme z nich vybrať najväčšie, čo bude naša hľadaná dĺžka.

Ako ale určíme čísla $D(x_1), D(x_2), ..., D(x_N)$? Postupne. Predpokladajme, že už máme vypočítané čísla $D(x_1), D(x_2), ..., D(x_{i-1})$ pre $i<N$, zaujíma nás vypočítať číslo $D(x_i)$. Všimnime si teraz štruktúru najdlhšej súvislej podpostupnosti končiacej v prvku x_i keďže hľadáme rastúcu súvislú podpostupnosť, platí:

$$D(x_i) = \begin{cases} x_{i-1} \geq x_i \Rightarrow D(x_i) = 1 \\ x_{i-1} < x_i \Rightarrow D(x_i) = D(x_{i-1}) + 1 \end{cases}$$

Toto nám postačuje na nájdenie našej dĺžky v čase $O(N)$. Všimnite si podobnosť štruktúry riešenia tejto úlohy s predchádzajúcou úlohou.

Problém batohu (Knapsack Problem)

Označme $O_{i,j}$ ako najväčšiu cenu vecí v batohu kapacity j takú, že si môžeme vybrať spomedzi vecí 1,...,i. Potom číslo $O_{N,M}$ prestavuje hodnotu vecí v najlepšom výbere vecí do batohu spomedzi všetkých vecí. Tieto čísla budeme po čítať postupne, predstavme si, že máme vypočítané všetky čísla pre i = 1,2,...,M: $O_{1,i}, O_{2,i}, ..., O_{k-1,i}$, ďalšie vypočítame nasledovne:

$$O_{k,i} = \max(O_{k-1,i}, \{O_{k-1,i-S_k} + C_k \mid S_k \geq i\}) \text{ pre } i = 1,2,...,M$$

Toto pozorovanie nám dáva návod na skonštruovanie riešenia pracujúceho v čase $O(NM)$.

Dominá(Dominoes – CEOI 1997, Nowy Sącz, Poland)

Najskôr si dominá transformujeme tak, že každému dominu bude prislúchať jedno číslo $x_i = |A_i\text{-}B_i|$. Potom $S = \sum_{i=1}^{N} x_i$ a $O_{i,j}$ označme ako minimálny počet preklopení (ak sa to nedá, tak $O_{i,j} = \infty$) niektorých domín

z množiny 1,...,i tak, že súčet hodnôt v hornom riadku na dominách 1,...,i po preklopení bude j. Keď už toto máme vypočítané, tak stačí nájsť medzi hodnotami $O_{N,i}$ taký súčet i, ktorý vieme nejakým otáčaním dosiahnuť a rozdiel $||S-i|-i|$ bude najmenší možný. Ak bude takýchto rozdielov viac (môže byť ešte jeden symetrický podľa $\frac{S}{2}$), tak vyberieme ten, ktorý preklopí oproti pôvodnému stavu menej domín.

Tieto čísla budeme po čítať postupne, predstavme si, že máme vypočítané všetky čísla pre i = 1,2,...,S: $O_{1,i}, O_{2,i}, ..., O_{k-1,i}$, ďalšie vypočítame nasledovne:
pre $i = 1,2,...,S$:

$$P_{k,i} = \{O_{k-1,i} \mid x_k \le 0\} \cup \{O_{k-1,i-x_k} \mid x_k > 0 \wedge x_k < i\}$$
$$Q_{k,i} = \{O_{k-1,i+x_k} + 1 \mid x_k < 0\} \cup \{O_{k-1,i} + 1 \mid x_k > 0\}$$
$$O_{k,i} = \min(P_{k,i}, Q_{k,i})$$

Kde $P_{k,i}$ korešponduje s možnosťou, že k-te domino neotočíme a $Q_{k,i}$ s možnosťou, že k-te domino otočíme. Toto pozorovanie nám dáva návod na skonštruovanie riešenia pracujúceho v čase $O(NS)$. Všimnite si podobnosť štruktúry riešenia s predchádzajúcim príkladom.

Plánovanie prednášok (Scheduling Lectures, ACM 1998, EastCentral NorthAmerica)

Označme $C_{i,j}$ ako najmenší počet prednášok, v ktorých sme schopný odučiť témy 1,...,i, pričom v poslednej použitej prednáške zostáva ešte j voľných minút. $D_{i,j}$ bude pre daný minimálny počet prednášok minimálna miera nespokojnosti študentov (bez miery nespokojnosti v poslednej ešte neuzatvorenej prednáške). Výsledok potom bude niektorý z ($C_{i,j}$,$D_{i,j}$) podľa toho, ktorý bude najlepší po uzatvorení poslednej prednášky.

Ako budeme uvedené čísla počítať? Postupne. Predstavme si že už máme vypočítané všetky čísla pre i = 0,1,2,...,L: $C_{1,i}, C_{2,i}, ..., C_{k-1,i}$, a $D_{1,i}, D_{2,i}, ..., D_{k-1,i}$, ďalšie vypočítame tak, že pre i = 0,1,2,...,L z ($C_{k-1,i}$, $D_{k-1,i}$) zohľadníme pridanie k-tej témy buď do aktuálnej prednášky, alebo uzatvorenie poslednej prednášky a vytvorenie novej. Toto robíme v duchu toho, aby si čísla $C_{i,j}$ a $D_{i,j}$ zachovali hore definovaný význam.

Táto myšlienka postačuje na vytvorenie algoritmu bežiaceho v čase $O(NL)$, ktorý nám určí najlepší výber prednášok.

Zber jabĺk (Apples, USACO, Fall 2002)

Pridáme si jedno jablko, ktoré spadne do bodu [0,0] v čase 0, toto jablko farmár určite chytí, a teda výsledok takto zmodifikovanej úlohy bude o jedna väčší ako pôvodnej úlohy. Jednotlivé jablká si označíme a_0, a_1, ..., a_N, pričom jablko a_0 je to naše umelo pridané. Označme s rýchlosť pohybu farmára, t_i, čas v ktorom padne jablko a_i a $d_{i,j}$ ako vzdialenosť dopadu jabĺk a_i a a_j. Definujme si potom nasledovné usporiadanie prvkov (jabĺk):

$$(a_i < a_j) \Leftrightarrow (\tfrac{d_{i,j}}{s} \le t_j - t_i)$$

Teda jablko a_i bude „menšie“ ako jablko a_j práve vtedy, keď platí, že ak by sa farmár po dopade jablka a_i hneď rozbehol k miestu dopadu jablku a_j, stihol by ho chytiť. Už nám stačí jablká len utriediť vzostupne podľa času dopadu a zostáva vyriešiť úlohu o najdlhšej vybranej rastúcej podpostupnosti z postupnosti jabĺk a_1, ..., a_N.

Palindromické cesty(Palindromic Tax Paths, USACO, Fall 2002)
Hľadáme počet všetkých palindromických ciest určitej dĺžky. Z axióm logiky vyplýva, že každá cesta, ktorú započítame musí niekde začínať a niekde končiť. Označme preto $C_{[A,B],[C,D],L}$ počet palindromických ciest dĺžky L, ktoré začínajú na súradniciach [A,B] a končia na súradniciach [C,D]. Naše hľadané číslo bude potom súčet všetkých $C_{[A,B],[C,D],L}$ takých, že $1 \leq A,B,C,D \leq N$. Všimnime si teraz štruktúru nejakej palindromickej cesty dĺžky $K \geq 2$. Prvý a posledný prvok tejto cesty a teda palindromická cesta dĺžky K, sa skladá s nejakej palindromickej cesty dĺžky K-2. Toto pozorovanie nám už dáva postup, ako vypočítať počet palindromických ciest dĺžky L.
Počty palindromických ciest budeme počítať postupne. Uvažujme, že už máme vypočítané počty ciest pre všetky $1 \leq A,B,C,D \leq N$: $C_{[A,B],[C,D],1}$, $C_{[A,B],[C,D],2}$, ..., $C_{[A,B],[C,D],K-1}$. Počet palindromických ciest dĺžky K je potom:

$$C_{[E,F],[G,H],K} = \begin{cases} [E,F] \neq [G,H] \Rightarrow C_{[E,F],[G,H],K} = 0 \\ [E,F] \neq [G,H] \Rightarrow C_{[E,F],[G,H],K} = \sum M_{[E,F],[G,H],K} \end{cases}$$

$$M_{[E,F],[G,H],K} = \{C_{[A,B],[C,D],K-2} \mid (|A-E|+|B-F|) = 1 \wedge (|C-G|+|D-H|) = 1\}$$

Toto nám dáva návod na skonštruovanie algoritmu bežiaceho v čase $O(LN^4)$.

Cesta okolo Kanady(Canada Tour, IOI 1993, Mendoza, Argentina)
Namiesto toho, aby sme hľadali najskôr cestu tam, potom späť, budeme hľadať tieto cesty naraz. Označme $O_{i,j}$ ako najväčší počet miest, ktoré môžeme navštíviť, ak pôjdeme nejakou cestou z mesta 1 do mesta i, pričom druhou (disjunktnou – neobsahuje žiadne rovnaké mesto) cestou pôjdeme z vrcholu 1 do vrcholu j. Hľadané číslo potom bude $O_{N,N}$. Čísla budeme počítať postupne tak, že vždy, keď budeme počítať číslo $O_{i,j}$, tak vieme, že už máme všetky čísla tvaru $O_{a,b}$ pre $a \leq i$ a $b \leq j$ už vypočítané. Pre $O_{i,j}$ ($i<j$) potom platí:

$$O_{i,j} = \max(\{O_{k,i}+1 \mid k<i \wedge \exists(k \leftrightarrow j)\} \cup \{O_{i,k}+1 \mid i<k<j \wedge \exists(k \leftrightarrow j)\})$$

Čo korešponduje s posunom jedného cestovateľa do oblastí, ktoré druhý cestovateľ ešte nevidel a teda to zaručuje disjunktnosť trás. Toto nám umožňuje zostrojiť algoritmus pracujúci v čase $O(N^2)$.

4.2.3. Dodatok k využitiu dynamického programovania
Ak chceme úlohu úspešne vyriešiť použitím dynamického programovania, vyžaduje si to, aby sme sa zamysleli nad štruktúrou optimálneho riešenia, optimálne riešenie definovali pomocou menších alebo jednoduchého rozšírenia menších riešení.

Všetko sa robí v duch počítania celkového riešenia od základov – od najmenších riešení – až po vrchol pyramídy – zadaná úloha.

4.2.4. Časové a pamäťové nároky
Pri riešení nie sú dôležité len požiadavky na čas výpočtu, ale aj na množstvo spotrebovaných pamäťových zdrojov. Vo väčšine prípadov, ale nie vždy, úlohy riešiteľné dynamickým programovaním majú zaujímavú vlastnosť, že pre veľkosť vstupu N, pamäťové nároky kladené na program sú N-krát menšie ako časové nároky.

Tento rozdiel, vzniká kvôli spôsobu konštrukcie väčšieho riešenia z menšieho. Postup k väčšiemu riešeniu si často vyžaduje prebrať nejakú množinu menších riešení, ktoré skombinujeme, alebo z ktorých si vyberieme jeden najlepší. Teda pre výpočet ďalšieho prvku, potrebujeme prebrať veľa (zhruba N) menších.