

2.2 Dynamické programovanie

Začnime s príkladom.

Príklad 2.2 *Uvažujme súťaž dvoch rovnako silných družstiev A, B , ktoré hrajú proti sebe zápasy. Pre každý zápas je rovnako pravdepodobné víťazstvo družstva A i B . Nech (i, j) je dvojica prirodzených čísel. Hovoríme, že družstvo A vyhralo súťaž s parametrami (i, j) , ak A dosiahne i víťazstiev skôr, ako B dosiahne j víťazstiev. Našou úlohou je vypočítať pravdepodobnosť $P(i, j)$ toho, že A vyhrá súťaž s parametrami (i, j) .*

Analýzou dostávame

- $P(0, j) = 1, j > 0$
- $P(i, 0) = 0, i > 0$
- $P(i, j) = \frac{1}{2}(P(i, j-1) + P(i-1, j)), i, j > 0^1$

Tento rekurentný vzťah navádza na použitie metódy rozdeľuj a panuj. Ak za veľkosť vstupu berieme $n = i + j$ a ako mieru zložitosti uvažujeme počet priradení a aritmetických operácií, potom metóda rozdeľuj a panuj vedie k zložitosti

$$T(1) = 1$$

$$\begin{aligned} T(n) &= 2T(n-1) + 3 = 2(2T(n-2) + 3) + 3 = 2^2T(n-2) + 2 \cdot 3 + 3 \\ &= 2^n + 3 \cdot \sum_{i=0}^{n-1} 2^i \end{aligned}$$

Lahko vidíme, že horný odhad je exponenciálny.

Čím je spôsobená táto veľká zložitosť?

- Vieme síce z riešenia menších podproblémov poskladať riešenie väčšieho podproblému, ale pri riešení sa vygeneruje exponenciálne veľa menších podproblémov menšieho rozsahu.
- Vieme, že je len polynomiálne veľa problémov menšieho rozsahu. To ale znamená, že niektoré sme počítali niekoľkokrát.

Pri metóde rozdeľuj a panuj sme veľký problém rozdelili na menšie, ktoré mohli byť riešené nezávisle. V prípade metódy dynamického programovania je tento princíp dotiahnutý do extrému:

- riešime postupne VŠETKY problémy (toho istého charakteru) MENŠIEHO rozsahu; postupujeme systematicky od menšieho rozsahu k väčšiemu
- riešenia si ukladáme k ďalšiemu použitiu
- používame, keď je počet problémov menšieho rozsahu rozumný

Nevýhodou je, že môžeme potrebovať veľkú pamäť.

Časté použitie tejto metódy je pre taký typ optimalizačných úloh, kde si riešenie možno predstaviť ako postupnosť nejakých rozhodnutí. V tejto situácii my hľadáme takú postupnosť rozhodnutí, ktorá niečo optimalizuje.

Abysme mohli použiť dynamické programovanie, musí sa optimalita dediť. Znamená to, že bez ohľadu na to, aké bolo prvé rozhodnutie, zvyšné rozhodnutia musia

tvoriť optimálne riešenie pre situáciu, ktorá nastala po prvom rozhodnutí. Inými slovami, ak súčasťou optimálneho riešenia väčšieho problému je riešenie menšieho problému, je toto riešenie optimálnym pre daný podproblém. *dedenie optimality*

Formálnejšie. Nech

- S_0 je situácia na začiatku.
- $\{r_1, \dots, r_k\}$ je množina rozhodnutí, ktoré môžeme urobiť v prvom kroku
- S_i je situácia po aplikovaní rozhodnutia r_i
- Γ_i je optimálna postupnosť rozhodnutí pre situáciu S_i

Hovoríme, že úloha (problém) spĺňa princíp dedenia optimality, ak o optimálnom riešení situácie S_0 platí, že vznikne výberom najlepšieho z riešení $r_1\Gamma_1, r_2\Gamma_2, \dots, r_k\Gamma_k$.

2.2.1 Súťaž dvoch družstiev

Použijeme na výpočet hodnoty $P(i, j)$ metódu dynamického programovania.

- $P(1, 0) = P(2, 0) = \dots = P(n, 0) = 0$
- $P(0, 1) = P(0, 2) = \dots = P(0, n) = 1$
- $P(i, j) = \frac{P(i-1, j) + P(i, j-1)}{2}$

Hodnoty budeme počítateľ postupne, začneme s rozsahom problému 1, potom prejdeme k rozsahu 2, ... Vypočítané hodnoty uložíme do tabuľky.

P(0,4)=1				
P(0,3)=1	P(1,4)			
P(0,2)=1	P(1,2)	P(2,2)		
P(0,1)=1	P(1,1)	P(2,1)	P(3,1)	
	P(1,0)=0	P(2,0)=0	P(3,0)=0	P(4,0)=0

Obrázok 2.1: Výpočet $P(i, j)$

Vzhľadom k závislosti väčšieho podproblému na menších je v tomto prípade zrejmé, že pri vyplňaní tabuľky môžeme postupovať po diagonálach, ktoré odpovedajú jednotlivým rozsahom problému.

Algoritmus 4 DP-1

```

1: for  $s \leftarrow 1$  to  $n$  do
2:    $P(0, s) \leftarrow 1; P(s, 0) \leftarrow 0;$ 
3:   for  $r \leftarrow 1$  to  $s - 1$  do
4:      $P(r, s) \leftarrow (P(r - 1, s) + P(r, s - 1))/2$ 

```

zložitosť

Analýzujeme teraz zložitosť algoritmu DP-1

čas Počítame $2 + 3 + \dots + (n + 1)$ hodnôt. Na výpočet každej z nich stačí konštantne veľa času. Celkovo teda máme $\mathbf{O}(n^2)$.

¹S pravdepodobnosťou $1/2$ vyhrá prvú hru A . Potom mu ostáva vyhrať $i - 1$ hier skôr, ako B vyhrá j hier. Analogicky, s pravdepodobnosťou $1/2$ vyhrá prvú hru B . Potom A musí vyhrať i hier skôr, ako B vyhrá $j - 1$ hier

pamäť všimnime si rozmer tabuľky

- tabuľka je veľkosti $O(n^2)$
- pri výpočte sa hodnoty zapisujú po diagonálach, pričom nová vzniká z informácií v starej; ľahko vidno, že stačí priestor dvoch diagonál
- pri lepšej manipulácii stačí len priestor jedinej diagonály²

2.2.2 Násobenie n matíc

Vstup: matice M_1, \dots, M_n , pričom rozmer matice M_i je $r_i \times s_i$, $r_{i+1} = s_i$. *problém*
 Výstup: matica $M = M_1 \times \dots \times M_n$

Zamyslime sa nad tým, čím môžeme ovplyviť zložitosť vypočítania tohto súčinu. Sú to

- efektívnosť algoritmu použitého na vynásobenie dvoch matíc
- poradie, v ktorom matice budeme násobiť

$$\begin{array}{ll} M_1 = 10 \times 20 & M_1 \times ((M_2 \times M_3) \times M_4) = 12200 \\ M_2 = 20 \times 5 & (M_1 \times M_2) \times (M_3 \times M_4) = 1550 \\ M_3 = 5 \times 100 & M_1 \times (M_2 \times (M_3 \times M_4)) = 800 \\ M_4 = 100 \times 1 & (M_1 \times (M_2 \times M_3)) \times M_4 = 31000 \end{array}$$

Ako vidíme, rôzne ozátvorkovanie/poradie násobenia má vplyv na výslednú zložitosť. Má preto zmysel najprv predvypočítať vhodné uzátvorkovanie vstupnej postupnosti matíc tak, aby sme pri ich násobení použili (pri fixovanom algoritme násobenia dvoch matíc) čo najmenší počet aritmetických operácií. Dostali sme sa takto k optimalizačnej úlohe, ktorá by sa mohla dať riešiť metódou dynamického programovania. Presvedčíme sa o tom :

- riešenie hľadáme ako postupnosť rozhodnutí – ktoré dve matice majú byť v tom ktorom kroku násobené?
- platí princíp dedenia optimality
 Nech M_{ik} označuje maticu, ktorá vznikla vynásobením $M_i \times M_{i+1} \times \dots \times M_k$.
 Nech A je optimálne poradie násobení, ktoré je také, že posledným násobením je násobenie $M_{1k} \times M_{k+1,n}$. Je zrejmé, že v A museli matice M_{1k} aj $M_{k+1,n}$ vzniknúť optimálnym násobením, lebo v opačnom prípade by sme ich výpočet v A nahradili poradím, ktoré by zmenšilo celkovú zložitosť.

Označme $cena(i, j)$ – počet operácií na získanie súčinu matíc $M_i \times M_{i+1} \times \dots \times M_j$. Potom zrejme

$$cena(i, j) = cena(i, k) + cena(k+1, j) + \text{zložitosť vynásobenia } M_{1k} \times M_{k+1,n}.$$

$$\underbrace{(M_i \times M_{i+1} \times \dots \times M_k)}_{M_{ik}} \times \underbrace{(M_{k+1} \times \dots \times M_j)}_{M_{k+1,j}}$$

Označme $m_{i,j}$ optimálny počet sledovaných operácií postačujúcich na získanie súčinu matíc $M_i \times M_{i+1} \times \dots \times M_j$ keď predpokladáme, že pri násobení dvoch matíc používame klasický školský algoritmus. Potom

$$m_{ij} = \begin{cases} \min_{i \leq k < j} \{(m_{ik} + m_{k+1,j} + r_i \times s_k \times s_j)\}, & i < j \\ 0, & i=j \end{cases}$$

²premýšľajte a naprogramujte

$m(1,4) = 800$ $k = 1$		
$m(1,3) = 6000$ $k = 2$	$m(2,4) = 600$ $k = 2$	
$m(1,2) = 1000$ $k = 1$	$m(2,3) = 10000$ $k = 2$	$m(3,4) = 500$ $k = 3$

Obrázok 2.2: Výpočet zátvorkovania

Uvedomme si, že m_{ij} je len optimálnym počtom operácií, potrebných na získanie výsledku M_{ij} . Nás ale zaujíma to, ktoré je to správne poradie násobenia matíc, pri ktorom dosiahneme optimálny počet operácií; potrebujeme do výrazu správne vložiť zátvorky. Preto si v tabuľke budeme pamätať nielen hodnotu m_{ij} , ale aj hodnotu $k(i, j)$, pri ktorej sa toto minimum m_{ij} dosahovalo. Hodnota $k(i, j)$ totiž určuje, za ktorú z matíc máme v danom násobení zložiť prvú pravú zátvorku.

$$\begin{aligned} (M_1 \times M_2 \times M_3 \times M_4), k(1,4) = 1 &\Rightarrow (M_1) \times (M_2 \times M_3 \times M_4) \\ k(2,4) = 2 &\Rightarrow (M_1) \times ((M_2) \times (M_3 \times M_4)) \end{aligned}$$

DÚ

Úloha:

- Napíšte program, ktorý pre vstupnú postupnosť rozmerov n matíc vypočíta, v akom poradí treba matice násobiť, aby pre daný algoritmus násobenia dvoch matíc bola výsledná zložitosť optimálna.
- Aká je časová a pamäťová zložitosť Vášho algoritmu?

2.2.3 Konštrukcia optimálneho binárneho vyhľadávacieho stromu

Keď používame binárny vyhľadávací strom (BVS) na reprezentáciu množiny, ktorú budeme často prezerat', má zmysel optimalizovať nielen čas na realizáciu jedného prístupu do BVS (teda zložitosť najhoršieho prípadu), ale v prípade, že poznáme (resp. odhadneme) početnosť prístupov do jednotlivých prvkov reprezentovanej množiny, môžeme skonštruovať BVS, ktorý bude minimalizovať čas, strávený prehľadávaním BVS počas celého výpočtu (čo je vlastne optimalizovanie zložitosti priemerného prípadu). Predpokladajme, že BVS má reprezentovať n -prvkovú množinu, pričom vieme, že početnosť prvku a_i je p_i . Bez ujmy na všeobecnosti budeme predpokladať, že prvky sú utriedené, teda $a_1 \leq a_2 \leq \dots \leq a_n$. Koľko vrcholov BVS T prezrieme, ak budeme p_1 razy hľadať prvok a_1 , p_2 razy prvok a_2 , ...?

$$c(T) = \sum_{i=1}^n h_i p_i$$

kde h_i je počet vrcholov v BVS T na ceste z koreňa do vrchola reprezentujúceho a_i . Je celkom prirodzené, že sa budeme snažiť konštruovať OBVS T , ktorý je optimálny vzhľadom na mieru $c(T)$.

Pri konštrukcii OBVS³ robíme v každom kroku rozhodnutie, ktorý z prvkov má byť v danom vrchole. Nech koreňom OBVS je prvok a_i . Potom ľavý (T_L) aj pravý (T_R) podstrom OBVS sú optimálnymi BVS pre množinu prvkov, ktorú reprezentujú. Preto je princíp dedenia optimality splnený a my môžeme použiť na konštrukciu

³OBVS konštruujeme postupne od koreňa smerom k listom

OBVS metódu dynamického programovania.

Čo vieme povedať o cene stromu T ? Platí

$$c(T) = c(T_L) + c(T_R) + \sum_{i=1}^n p_i$$

nakoľko každá z ciest sa v porovnaní so stromom T_L , resp. T_R predĺžila o 1.

- nech $cena(i, j)$ označuje cenu optimálneho BVS pre množinu prvkov a_i, a_{i+1}, \dots, a_j
- nech $cena(i, i-1) = 0$
- potom

$$cena(i, j) = \min_{i \leq k \leq j} \{cena(i, k-1) + cena(k+1, j) + \sum_{k=i}^j p_k\}$$

- kvôli možnosti konštrukcie OBVS potrebujeme poznať aj hodnotu $k(i, j)$, pri ktorej sa dosahuje $cena(i, j)$.

Úloha:

- Naprogramujte popísaný algoritmus a analyzujte jeho časovú a priestorovú zložitosť. *DÚ*
- Ako sa zmení uvedený algoritmus ak budeme predpokladať, že navyše poznáme aj početnosť vyhľadávania prvkov, ktoré sa v množine, reprezentovanej BVS, nenachádzajú? Teda keď poznáme $q(i)$ – početnosť vyhľadávania prvku, ktorý je medzi a_{i-1} a a_i . Pritom predpokladáme, že $q(1)$ je početnosť vyhľadávania prvkov menších ako a_1 , $q(n+1)$ je početnosť vyhľadávania prvkov väčších ako a_n .

2.2.4 0/1 Plnenie batoha (0/1 Knapsack problem)

Uvažujme nasledujúci problém. Daných je n objektov s váhami w_i a batoh kapacity M . Ak umiestnime do batoha objekt w_i , získame profit p_i . Našou úlohou je naplniť batoh tak, aby sme získali čo možno najväčší zisk. Formálne: *problém*

Vstup: $w_i, p_i, M, 1 \leq i \leq n$

Výstup: $\max \sum_{i=1}^n x_i p_i$, pričom $\sum_{i=1}^n x_i w_i \leq M$ a $x_i \in \{0, 1\}$

Aby sme mohli na riešenie použiť metódu dynamického programovania, musíme riešenie vyjadriť ako postupnosť rozhodnutí a overiť platnosť dedenia optimality.

Označme $KNAP(a, b, Y)$ problém 0/1 plnenia batoha pri vstupoch $w_i, p_i, a \leq i \leq b, M = Y$. Zrejme pôvodný problém 0/1 plnenia batoha je $KNAP(1, n, M)$. Za rozhodnutie možno považovať zaradenie, resp. nezaradenie objektu do batohu.

Nech y_1, y_2, \dots, y_n je optimálne priradenie hodnôt premenným x_1, x_2, \dots, x_n .

Ak $y_1 = 0$, tak y_2, \dots, y_n musí byť optimálne riešenie $KNAP(2, n, M)$.

Ak $y_1 = 1$, tak y_2, \dots, y_n musí byť optimálne riešenie $KNAP(2, n, M - w_1)$.

Označme $g_j(y)$ hodnotu $KNAP(j+1, n, y)$. Zrejme $g_0(M)$ je hodnota optimálneho riešenia $KNAP(1, n, M)$ a

$$g_0(M) = \max\{g_1(M), g_1(M - w_1) + p_1\} \quad (*)$$

Nech y_1, y_2, \dots, y_n je optimálne riešenie pre $KNAP(1, n, M)$. Potom $\forall j, 1 \leq j \leq n$,

- y_1, \dots, y_j je optimálne riešenie pre $KNAP(1, j, \sum_{1 \leq i \leq j} w_i y_i)$
- y_{j+1}, \dots, y_n je optimálne riešenie pre $KNAP(j+1, n, M - \sum_{1 \leq i \leq j} w_i y_i)$

Preto môžeme (*) zovšeobecniť:

$$g_i(y) = \max\{g_{i+1}(y), g_{i+1}(y - w_{i+1}) + p_{i+1}\} \quad (**)$$

Ak si uvedomíme, že $g_n(y) = 0 \forall y$, môžeme postupne počítat' $g_{n-1}(y), \dots$. Tento postup vlastne odpovedal postupnému priradovaniu hodnôt x_n, x_{n-1}, \dots

Nemusíme postupovať odzadu, ale sa dá aj spredu. Ak označíme $f_j(X)$ hodnotu optimálneho riešenia pre $KNAP(1, j, X)$, tak

$$\begin{aligned} f_n(M) &= \max\{f_{n-1}(M), f_{n-1}(M - w_n) + p_n\}, \text{ resp.} \\ f_i(X) &= \max\{f_{i-1}(X), f_{i-1}(X - w_i) + p_i\}, \text{ čo súčasne s faktom } f_0(X) = 0 \text{ a} \\ & f_i(X) = -\infty, X < 0 \text{ dáva návod, ako postupne počítat' } f_1, f_2, \dots \end{aligned}$$

Úloha:

DŮ

- Domyslite detaily a naprogramujte riešenie problému 0/1 plnenia batoha.
- Nech C je matica cien ohodnoteného grafu s vrcholmi $1, 2, \dots, n$; $C[i, j]$ udáva kladnú dĺžku hrany medzi vrcholmi i a j . Použite metódu dynamického programovania na výpočet matice S dĺžok najkratších ciest; $S[i, j]$ bude cena najkratšej cesty z vrchola i do vrchola j , pričom pod dĺžkou cesty rozumieme súčet cien hrán, ktoré cestu tvoria.
- Uvažujme problém hľadania *najdlhšej spoločnej podpostupnosti* dvoch postupností $X = X_1, \dots, X_n$, $Y = Y_1, \dots, Y_m$, ktorá vznikne vynechaním niektorých znakov z každej z nich. Označme $LCS(i, j)$ dĺžku najdlhšej spoločnej podpostupnosti postupností X_1, \dots, X_i a Y_1, \dots, Y_j ; zaujíma nás zrejme $LCS(n, m)$. (LCS(ABAKUS, AUTOBUS)=4; najdlhšia spoločná podpostupnosť je ABUS) Základom algoritmu pre výpočet $LCS(n, m)$ založenom na dynamickom programovaní je nasledujúci vzťah:

$$LCS(i, j) = \begin{cases} 1 + LCS(i-1, j-1), & \text{ak } X_i = Y_j \\ \min\{LCS(i-1, j), LCS(i, j-1)\}, & \text{inak} \end{cases}$$

Napište program, ktorý nielen vypočíta $LCS(n, m)$, ale aj príslušnú najdlhšiu spoločnú podpostupnosť. Aká je zložitosť Vášho programu?

- Majme ohodnotený graf $G = (V, E)$, zadaný maticou cien: $C[i, j]$ je cena hrany z vrchola i do vrchola j . Našou úlohou je nájsť takú permutáciu π vrcholov $2, 3, \dots, n-1$, ktorá minimalizuje dĺžku kružnice $1, \pi, 1$; pod cenou kružnice rozumieme súčet cien jej hrán. Problému sa hovorí problém obchodného cestujúceho (TSP)

Nech $S = \{1, 2, \dots, n\}$, $j \in S$. Označme $C(S, j)$ dĺžku najkratšej cesty $1\pi j$, kde π je permutácia množiny $S - \{1, j\}$.

Základom riešenie problému TSP dynamickým programovaním je nasledujúci vzťah

$$C(S, j) = \min_i \{C(S - \{j\}, i) + C[i, j]\}$$

Napište program, ktorý rieši TSP dynamickým programovaním. Aká je zložitosť Vášho programu?

2.3 Greedy algoritmy

Je to jedna z veľmi rozšírených metód, ktorá sa väčšinou používa pri riešení optimalizačných úloh. Tieto úlohy často možno charakterizovať nasledovne:

- Vstupom je n -prvková množina, resp. postupnosť
- Cieľom je vytvoriť podmnožinu (podpostupnosť), ktorá spĺňa nejaké podmienky (odrážajúce charakter problému). Každé riešenie, ktoré spĺňa tieto ohraničenia, sa nazýva *prípustné riešenie*.
- Navyše je daná je účelová funkcia, definovaná na množine prípustných riešení.

Výstupom je to prípustné riešenie, ktoré optimalizuje účelovú funkciu. Takéto riešenie voláme *optimálne riešenie*.

Snažíme sa nájsť *rýchly* algoritmus, ktorý generuje priamo optimálne riešenie. Greedy metóda vedie k algoritmom, ktoré pracujú v taktach. V každom takte sa vyberie najvhodnejší kandidát na zaradenie do optimálneho riešenia. Ak vybraný kandidát nemôže byť do riešenia zaradený (nespĺňa podmienky prípustného riešenia, dokázateľne nemôže byť súčasťou optimálneho riešenia, ...), definitívne sa vyradí z množiny potenciálnych kandidátov. Pri výbere kandidátov zväčša aplikujeme "pažravý" prístup - snažíme sa zohľadňovať optimalizačné kritérium.

Algoritmus 5 Greedy(A,n)

```

1: riešenie  $\leftarrow \emptyset$ ;
2: for  $i=1$  to  $n$  do
3:    $X \leftarrow SELECT(A)$ ;
4:   if PRÍPUSTNÉ(riešenie, X) then riešenie  $\leftarrow UNION(riešenie, X)$ 
5: return(riešenie)

```

Uvedomme si, čo ovplyvňuje úspešnosť takejto metódy

- SELECT – má vplyv nielen na zložitosť, ale i na to, či získané riešenie bude optimálne
- PRÍPUSTNÉ – predpokladáme, že vieme rozhodnúť, či daný prvok bude súčasťou (optimálneho) riešenia. Zrejme ovplyvňuje nielen zložitosť, ale aj kvalitu.

Príklad 2.3 *Daná je množina hodnôt mincí a suma, ktorú treba zaplatiť. Hodnotou mince účelovej funkcie je počet mincí, ktorý sa snažíme minimalizovať.*

Greedy metóda vedie k nasledovnému postupu — začni s bankovkami s maximálnou možnou hodnotou a plať, kým sa dá. Potom vezmi bankovky s menšou hodnotou a plať, kým sa dá,...

Ak vezmeme ako základnú množinu mincí 1,2,5,10,..., vedie greedy metóda k optimálnemu riešeniu. Čo však, ak vezmeme ako množinu mincí 1,9, 11, a suma, ktorú potrebujeme zaplatiť, je 19?

Greedy algoritmus vedie k prípustnému riešeniu – 11,1,1,1,1,1,1,1. Optimálne riešenie je však 9,9,1.

Úloha: Pre aké hodnoty mincí vedie greedy metóda k optimálnemu riešeniu? DÚ

2.3.1 Plnenie batoha (s racionálnymi koeficientami)

Uvažujme nasledovný problém

problém

Vstup: n objektov určených váhou a ziskom a kapacita batoha
 M kapacita batoha
 w_i váha i -teho objektu
 p_i profit; ak prenesieme časť x_i objektu i , $0 \leq x_i \leq 1$, tak získame cenu (profit) $p_i x_i$

Úloha Naplniť batoh tak, aby sme maximalizovali zisk $Z = \sum_{i=1}^n p_i x_i$, pričom

1. $\sum_{i=1}^n w_i x_i \leq M$
2. $0 \leq x_i \leq 1$, $p_i, w_i > 0$, $1 \leq i \leq n$

Prípustným je každé také riešenie (x_1, x_2, \dots, x_n) , ktoré spĺňa (1.) a súčasne (2.) Optimálnym riešením je to prípustné riešenie, ktoré maximalizuje zisk Z .

Príklad 2.4 *Majme vstup*

$$n = 3, M = 20, (p_1, p_2, p_3) = (25, 24, 15), (w_1, w_2, w_3) = (18, 15, 10)$$

<i>Prípustné riešenia:</i>	$\sum_{i=1}^n w_i x_i$	$\sum_{i=1}^n p_i x_i$
1. $(1/2, 1/3, 1/4)$	16.5	24.25
2. $(1, 2/15, 0)$	20	28.2
3. $(0, 2/3, 1)$	20	31
4. $(0, 1, 1/2)$	20	31.5

Zrejme ak $\sum_{i=1}^n w_i \leq M$, tak položíme $x_i = 1$ a máme optimálne riešenie. Preto predpokladajme, že $\sum_{i=1}^n w_i > M$.

Existuje viacero greedy stratégií, ktoré môžeme skúsiť použiť:

1. plníme podľa maximálneho zisku, ktorý by priniesol celý objekt – nie vždy dáva optimálne riešenie
2. podľa najmenšej váhy
3. podľa maximálneho relatívneho profitu p_i/w_i – toto vedie k optimálnemu riešeniu

Všimli sme si, že sme mohli brať do úvahy tri miery – *váhu*, *profit*, *relatívny zisk* – podľa toho sa menilo poradie, v ktorom sme objekty skúmali. Pritom len jeden z prístupov vedie k optimálnemu riešeniu.

korektnosť greedy pre batoh **Fakt 2.3** *Nech $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \frac{p_3}{w_3} \leq \dots \leq \frac{p_n}{w_n}$. Potom greedy algoritmus založený na maximálnom relatívnom zisku dáva optimálne riešenie.*

Dôkaz: Majme

$$\begin{aligned} X &= (x_1, x_2, \dots, x_n) && \text{riešenie algoritmu greedy} \\ Y &= (y_1, y_2, \dots, y_n) && \text{optimálne riešenie} \end{aligned}$$

príčom predpokladáme, že sú rôzne.

Ukážeme, ako modifikáciami optimálneho riešenia, ktoré nezhoršujú jeho kvalitu, transformujeme toto optimálne riešenie na greedy riešenie. Z toho bude vyplývať, že greedy riešenie je optimálne.

Nech j je minimálny index taký, že $x_j \neq 1$. Ak X nie je optimálne riešenie, tak pre optimálne riešenie Y platí $\sum p_i x_i < \sum p_i y_i$ a navyše môžeme predpokladať, že $\sum w_i x_i = M$. Nech k je minimálny index taký, že $x_k \neq y_k$.

$$\begin{array}{ll}
k < j & x_k = 1, y_k < 1 \Rightarrow x_k > y_k \\
k = j & \Rightarrow x_k > y_k \\
j < k & \text{nie je možné, lebo by sme dostali } \sum w_i y_i > M
\end{array}$$

Preto $\mathbf{x}_k > \mathbf{y}_k$. Nech $Z = (z_1, z_2, \dots, z_n)$ je riešenie, ktoré získame nasledovnou úpravou optimálneho riešenia:

$$\begin{array}{ll}
z_i = x_i, & i \leq k; \\
z_i, & \sum_{k < i \leq n} w_i (y_i - z_i) = w_k (z_k - y_k).
\end{array}$$

Dostávame

$$\begin{aligned}
\sum p_i z_i &= \sum p_i y_i + \frac{(z_k - y_k) w_k p_k}{w_k} - \sum_{k < i \leq n} (y_i - z_i) \frac{w_i p_i}{w_i}, \quad \frac{p_i}{w_i} \leq c \quad \Rightarrow \\
&\geq \sum p_i y_i + [(z_k - y_k) w_k - \sum_{k < i \leq n} (y_i - z_i) w_i] \frac{p_k}{w_k} \\
&= \sum p_i y_i
\end{aligned}$$

- Ak $X = Z$, vtedy je X optimálne
- Ak $Z \neq X$, vtedy $X \leftarrow Z$ a aplikuj uvedený postup. Po konečnom počte krokov musí nastať situácia, že $X = Z$.

2.3.2 Úlohy s terminovaním

Majme n úloh a jeden procesor. O každej úlohe u_i vieme, dokedy musí byť spočítaná (deadline $d(i)$) a aký zisk (profit $p(i)$) budeme mať, ak ju načas zrátame. Predpokladáme, že každá úloha sa počíta jednotku času.

Úloha: Vybrať podmnožinu úloh, ktoré všetky môžu byť spočítané načas tak, aby sme maximalizovali zisk.

Prípustné riešenie: Podmnožina J úloh (resp. ich indexov), ktoré sa dajú spracovať pred deadline (vrátane)

Hodnota súčet získaných profitov

Optimum: maximálna hodnota

Príklad 2.5 *Majme*

	p_i	d_i
1	100	2
2	10	1
3	15	3
4	27	1

prípustné riešenie	postup	zisk
1,2	2,1	110
1,3,2	2,1,3	125
1,4,3	4,1,3	142

Ako máme hľadať optimálne riešenie? Jednou možnosťou je pre všetky podmnožiny úloh overiť, či sa dajú usporiadať tak, aby bola každá vypočítaná načas a ak áno, spočítať získaný zisk. Potom stačí vybrať tú podmnožinu, ktorá prináša najväčší zisk. Takéto riešenie ale vedie k exponenciálnej zložitosti.

Použijeme preto greedy metódu a dokážeme, že dáva optimálne riešenie. K popisu stačí určiť, akú stratégiu na výber kandidáta volíme a ako rozhodujeme o tom, či je daný kandidát vhodný:

SELECT

- Vyberáme prvok, ktorý má maximálny profit. Pri rovnosti profitov zachováme relatívne poradie zo vstupu.

PRÍPUSTNÉ

Ako overiť, či skúmaná podmnožina indexov odpovedá takej množine úloh, ktoré sa dajú zrátavať načas? Prezeranie všetkých usporiadaní je neefektívne. Ľahko vidno, že ak má byť úloha dopočítaná v čase t , musí byť nanajvyš $t - 1$ úloh počítaných pred ňou.

- Majme nejaké poradie úloh π . Ak pre každú úlohu je poradové číslo úlohy v π nanajvýš rovné jej deadline, dajú sa úlohy spočítať v poradí π a všetky budú načas. Preto usporiadame úlohy podľa deadlineov neklesajúco, pričom pri rovnosti deadlineov zachováваме relatívne poradie zo vstupu.
- Potom v čase $O(n)$ vieme overiť, či $\text{Prípustné}(\text{Riešenie}, X)$ dáva hodnotu TRUE.

V našom prípade:

Riešenie = {1}
 Riešenie = {1}, $X = 4$, $\text{Prípustné}(1,4) = \text{True} \Rightarrow \text{Riešenie} = \{1,4\}$
 Riešenie = {1,4}, $X = 3$, $\text{Prípustné}(1,4,3) = \text{True} \Rightarrow \text{Riešenie} = \{1,4,3\}$
 Riešenie = {1,4,3}, $X = 2$, $\text{Prípustné}(1,4,3,2) = \text{False} \Rightarrow \text{Riešenie} = \{1,4,3\}$

Odpovedajúci postup: 4,1,3

korektnosť greedy riešenia pre úlohy s terminovaním

Dôkaz: Musíme vyargumentovať optimalitu riešenia, získaného greedy metódou. Nech $G = \{g_1, \dots, g_k\}$ je riešenie získané greedy metódou $O = \{o_1, \dots, o_\ell\}$ je optimálne riešenie.

Predpokladáme

- G, O sú usporiadané nerastúco podľa profitov
 $p(g_1) \geq p(g_2) \geq \dots \geq p(g_k)$ a $p(o_1) \geq p(o_2) \geq \dots \geq p(o_\ell)$
- ak dve úlohy majú rovnaký profit, tak je zachované relatívne poradie zo vstupu
 $p(i_q) = p(i_{q+1}) \Rightarrow i_q < i_{q+1}$

Majme $G \neq O$

$G \subset O$ V tomto prípade optimálne riešenie O obsahuje úlohu u , ktorú neobsahuje greedy riešenie G . Keďže greedy metódou bola každá úloha uvažovaná ako potenciálny kandidát na zaradenie do G , bola zvažovaná aj úloha u . Jej nezaradenie do G znamená, že jej pridanie k G znemožnilo zráťania úloh načas. To znamená, že by tomu muselo byť tak aj v prípade riešenia O . Preto **nemôže nastať**

$O \subset G$ Tento fakt by implikoval, že O nie je optimálne riešenie. Preto **nemôže nastať**

Z uvedeného vyplýva, že **existuje a (minimálne také), že $g_a \neq o_a$** . Ukážeme, že ak zaradíme g_a do O , budeme vedieť vylúčiť z O také o_s , že takouto zámennou nezhoršíme kvalitu riešenia O a zachováme jeho riešiteľnosť. Po konečnom počte takýchto krokov skončíme v situácii, keď $O = G$, čo bude dokazovať optimalitu riešenia G .

- môže patriť g_a do O ? Nie, pretože:
 ak $p(g_a) > p(o_a)$, tak ďalej sú už len úlohy s menším profitom
 ak $p(g_a) = p(o_a)$, tak ak $g_a < o_a$
- ako hľadať o_s ?
 o_s odpovedá prvej úlohe spomedzi úloh s indexom $o_t, t \geq a$, ktorá má najmenší deadline

Keďže $p(g_a) \geq p(o_a) \geq p(o_s)$, kvalitu sme nepokazili. Ostáva ukázať, že sme zachovali riešiteľnosť. Inými slovami, že $O' = O \setminus \{o_a\} \cup \{o_s\}$ je množina indexov odpovedajúca úlohám, ktoré sa dajú spočítať tak, že každá z nich je spočítaná načas.

Preusporiadajme úlohy O podľa deadlinov nerastúco. Získame postupnosť α, o_s, β . Vzhľadom k voľbe indexu s je zrejme, že všetky úlohy s indexom z α sa nachádzajú aj v riešení G získanom greedy metódou. Existuje preto usporiadanie α' postupnosti indexov α, o_s , ktoré odpovedá nerastúcim deadlineom. Navyše, ak by sme úlohy riešili v tomto poradí, bude každá z nich vyriešená načas. Preto

$$\alpha', g_a, \beta$$

je poradie, v ktorom je každá z úloh s indexom z O' spočítaná načas.

Úloha: Zamyslite sa nad implementáciou a zložitouťou.

DÚ

2.3.3 Optimálne zlučovanie súborov

Majme n utriedených súborov $F(1), F(2), \dots, F(n)$. Úlohou je napísať zlučovací algoritmus na vytvorenie jediného utriedeného súboru F . Pritom vieme, že pre zlučovanie dvoch utriedených súborov poznáme efektívny algoritmus (lineárnej zložitosti). Vzhľadom k tomu, že porovnanie dvoch prvkov je zanedbateľné vzhľadom k prístupu do súboru, je prirodzenou mierou zložitosti celkový počet prístupov k prvkom. Presnejšie: $\sum p(i)|F(i)|$, kde $|F(i)|$ je počet prvkov v súbore $F(i)$ a $p(i)$ je počet zlučovaní, v ktorých sa zúčastňujú prvky pôvodne uložené v súbore $F(i)$.

Majme nejaký zlučovací algoritmus. K nemu možno priradiť binárny zlučovací strom nasledovným spôsobom:

- listy stromu reprezentujú súbory, ktoré máme zlučovať
- vnútorné vrcholy reprezentujú súbory, ktoré vznikajú v priebehu zlučovania; $F(i)$ je súbor, ktorý vznikol v $(i - n)$ -tom zlučovaní, $i > n$.

Ľahko vidno, že k jednému algoritmu je zlučovací strom priradený jednoznačne až na izomorfizmus. Na druhej strane jeden zlučovací strom reprezentuje viacero algoritmov.

Zrejme zložitost' algoritmu možno vyjadriť ako $\sum h(i) \cdot |F(i)|$, kde $h(i)$ je hĺbka listu reprezentujúceho súbor $F(i)$; $\sum h(i) \cdot |F(i)|$ sa volá vážená dĺžka ciest. Optimálny zlučovací algoritmus je taký, pre zlučovací strom ktorého je $\sum h(i) \cdot |F(i)|$ minimálne.

Riešme greedy metódou.

SELECT

V každom kroku zlučuj dva súbory s najmenšou veľkosťou.

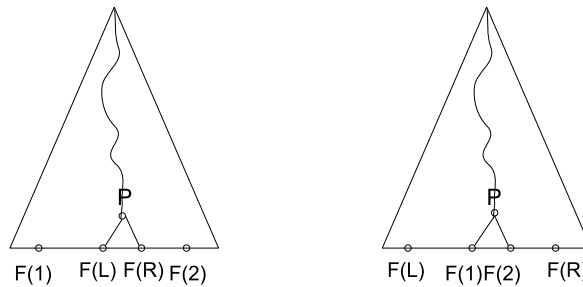
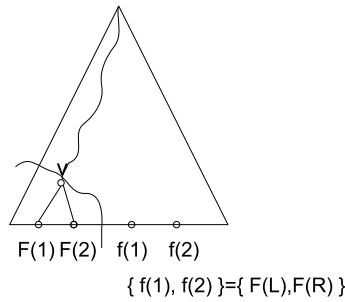
Kvôli *korektnosti* predpokladajme, že máme dva stromy - TO (optimálny zlučovací algoritmus) a TG (greedy). Indukciou ukážeme, že z hľadiska váženej dĺžky ciest zlučovacieho stromu je TG rovnako dobrý ako TO . *korektnosť*

Pre triviálne prípady $n = 1, 2$ tvrdenie zrejme platí

Báza indukcie

Nech tvrdenie platí pre zlučovanie m súborov, $m < n$.

IP



Indukčný krok:

- Nech $|F(1)| \leq |F(2)| \leq \dots \leq |F(n)|$. Po prvom kroku greedy algoritmu sa vytvorí nový vrchol v so synmi $F(1), F(2)$.
- Nech v zlučovacom strome TO je P vnútorný vrchol najďalej od koreňa. Nech $F(L), F(R)$ sú súbory reprezentované ľavým, resp. pravým synom vrchola P . Nech $f(1), f(2)$ sú vrcholy stromu TO reprezentujúce súbory $F(1), F(2)$. Spravme v strome TO zmenu tak, aby sa $f(1)$ stalo ľavým synom vrchola P , $f(2)$ pravým synom vrchola P . Tým sme dostali nový zlučovací strom NTO .

Všimnime si, ako sa zmenila cena $c(NTO)$ vážených ciest stromu NTO oproti cene $c(TO)$ stromu TO .

$$c(NTO) - c(TO) = h(L)F(L) + h(R)F(R) + h(1)F(1) + h(2)F(2) - \{h(L)F(1) + h(R)F(2) + h(1)F(L) + h(2)F(R)\}$$

Keďže

$$h(L) \leq h(1), \quad h(R) \leq h(2), \quad F(1) \leq F(L), \quad F(2) \leq F(R)$$

je $c(NTO) - c(TO) \leq 0$. Oba stromy TG aj na NTO sú zlučovacie stromy pre zlučovanie $n-1$ súborov $F(1)+F(2), F(3), \dots, F(n)$. Preto je podľa IP cena TG rovnaká ako cena NTO a teda greedy algoritmus dáva optimálne riešenie.

□

2.3.4 Minimálna kostra.

Posledným príkladom použitia greedy metódy bude algoritmus konštrukcie minimálnej kostry v grafe.

problém

Daný je súvislý neorientovaný ohodnotený graf. Kostra je podgraf tohto grafu, ktorý obsahuje všetky vrcholy a tvorí strom. Cena kostry je súčet cien priradených hranám, ktoré tvoria kostru. Chceme skonštruovať kostru s minimálnou cenou.

riešenie

SELECT: sú dva základné princípy

PRIM: vyber hranu s minimálnou cenou tak, aby čiastočné riešenie tvorilo strom

KRUSKAL: vyber hranu s minimálnou cenou tak, aby čiastočné riešenie tvorilo les

Ukážeme *korektnosť* Kruskalovho algoritmu:

korektnosť

Nech K je kostra vytvorená Kruskalovým algoritmom, O optimálna kostra. Označme

$E(K)$ množinu hrán kostry K

$E(O)$ množinu hrán kostry O .

Ak $E(O) = E(K)$, potom majú obe kostry rovnakú cenu. Predpokladajme teda, že $\mathbf{E}(O) \neq \mathbf{E}(K)$ a uvažujme hranu $e \in E(K) \setminus E(O)$ s *minimálnou* cenou. Pridanie hrany e k $E(O)$ spôsobí vznik jediného cyklu $e, e(1), e(2), \dots, e(k)$ v O . Zrejme aspoň jedna z hrán $e(1), e(2), \dots, e(k)$ nepatrí do $E(K)$ ⁴. Nech je to hrana $e(j)$. Potom $\text{cena}(e(j)) \geq \text{cena}(e)$ (ak by to neplatilo, bol by Kruskalov algoritmus uvažoval hranu $e(j)$ skôr ako hranu e .)

Zmena $E(O) \leftarrow E(O) \setminus \{e\} \cup \{e(j)\}$ nezväčší cenu takto vzniknutej kostry NO . Pritom

$$|E(K) \setminus E(NO)| < |E(K) \setminus E(O)|$$

Takto postupnými modifikáciami, ktoré nezhoršujú cenu, dostaneme z kostry O kostru K .

□

2.3.5 *Matroidy—teoretické základy pre greedy metódy

⁵V tejto časti sa budeme venovať teórii, ktorá nám môže pomôcť rozhodnúť, či je daná úloha vhodná na použitie greedy metódy alebo nie. Táto metóda je založená na tzv. matroidoch. Je to len ukážka teórie, nepokrýva všetky úlohy, ktoré sú riešiteľné pomocou greedy metódy.

Definícia 2.1 Matroid je usporiadaná dvojica $M = (S, \mathcal{I})$, ktorá spĺňa nasledujúce matroidy podmienky:

1. Sje neprázdna konečná množina
2. Ije taký neprázdny systém podmnožín S , tzv. nezávislé podmnožiny množiny S , pre ktoré $(B \in \mathcal{I} \wedge A \subseteq S) \Rightarrow A \in \mathcal{I}$. Hovoríme, že M má vlastnosť dedičnosti. Samozrejme, \emptyset musí patriť do \mathcal{I} .
3. Ak $A \in \mathcal{I}, B \in \mathcal{I}, |A| < |B|$, tak existuje $x \in B - A$ tak, že $A \cup \{x\} \in \mathcal{I}$. Hovoríme, že M má vlastnosť zameniteľnosti.

Príklad 2.6 Príkladom matroidu je tzv. grafový matroid $M_G = (S_G, \mathcal{I}_G)$, ktorý získame z neorientovaného grafu $G = (E, V)$:

- $S_G = E$
- ak A je podmnožina hrán E tak $A \in \mathcal{I}_G \Leftrightarrow A$ je acyklická. Inými slovami, podmnožina hrán je nezávislá práve vtedy, ak tvorí les.

Overíme, že M_G je matroid.

1. $S_G = E$ je konečná neprázdna množina.

⁴prečo?

⁵Prebraté z Cormen, Leiserson, Rivest: Introduction to Algorithms; časť 17.4

2. \mathcal{I}_G má vlastnosť dedičnosti, pretože podmnožina acyklického grafu je acyklická.
3. Keď máme dva lesy $A, B, |B| > |A|$, tak les B má menej stromov ako les A . Preto les B obsahuje taký strom T , ktorý má v dvoch rôznych stromoch lesa A . Dokonca viac - vieme nájsť takú hranu (u, v) v T_B , že u, v ležia v rôznych stromoch lesa A . Preto (u, v) môžeme pridať k A a žiaden cyklus nevznikne; preto $A \cup (u, v) \in \mathcal{I}_G$.

rozšírenie

Nech $M = (\mathcal{S}, \mathcal{I})$ je matroid, $A \in \mathcal{I}$. Prvok $x \notin A$ nazveme *rozšírenie* A , ak $A \cup \{x\} \in \mathcal{I}$.

Nezávislá podmnožina A v matroide M nazveme *maximálnou* ak nemá rozšírenie.

Veta 2.4 *Všetky maximálne nezávislé podmnožiny v matroide sú rovnako veľké.*

Dôkaz: Argumentujme sporom. Nech A, B sú maximálne nezávislé, $|B| > |A|$. Potom existuje $x \in B - A$: $A \cup \{x\} \in \mathcal{I}$, čo je spor s maximalitou. \square

vážený matroid

Ak k matroidu $M = (\mathcal{S}, \mathcal{I})$ pridáme váhovaciu funkciu w , ktorá $\forall x \in \mathcal{S}$ priradí kladnú váhu $w(x)$, dostaneme vážený matroid/matroid s váhou.

Mnoho optimalizačných úloh možno formulovať ako hľadanie najťažšej maximálnej nezávislej podmnožiny vo váženom matroide. Ukážeme, že všetky tieto optimalizačné úlohy môžeme riešiť greedy metódou.

Príklad 2.7 *príkladom takejto optimalizačnej úlohy je hľadanie najlacnejšej kostry v ohodnotenom grafe $G = (V, E, c)$, kde c je funkcia, ktorá každej hrane priradí nezápornú cenu. Keďže hľadanie minimálnej kostry⁶ je minimalizačná úloha, pretransformujeme ju vhodnou definíciou váhovacej funkcie na maximalizačnú. Namiesto ceny c budeme uvažovať váhu w , ktorá hrane $e \in E$ priradí váhu $w(e) = c_{max} - c(e)$, kde c_{max} je číslo, ktoré je väčšie ako maximálna cena hrany v E .*

Zrejme:

- $w(e) > 0$ pre všetky hrany $e \in E$
- maximálna nezávislá podmnožina A v matroide je kostra
- $w(A) = (|V| - 1)c_0 - w(A)$

Postúpme k metóde greedy. $M = (\mathcal{S}, \mathcal{I})$ naďalej označuje matroid s váhovacou funkciou w . Všeobecný algoritmus použijeme greedy metódy na vážených matroidoch možno vyjadriť Algoritmom 6 GREEDY(M, w).

Algoritmus 6 GREEDY(M, w)

- 1: $A \leftarrow \emptyset$
 - 2: **for** $x \in \mathcal{S}$ nerastúco podľa $w(x)$ **do**
 - 3: **if** $A \cup \{x\} \in \mathcal{I}$ **then** $A \leftarrow A \cup \{x\}$
 - 4: **return** A
-

Korektnosť tohto prístupu dokážeme v nasledujúcich lemach.

vlastnosť greedy-volby

Lema 2.5 *Nech v matroide $M = (\mathcal{S}, \mathcal{I})$ s váhovacou funkciou w je x najťažší prvok v \mathcal{S} taký, že x je nezávislý. Ak také x existuje, potom existuje optimálne riešenie A , ktoré obsahuje x .*

⁶Minimálna kostra v grafe G je taký podstrom T , ktorý obsahuje všetky vrcholy a pre ktorý je súčet cien hrán minimálny.

Dôkaz: Ak také x neexistuje, potom je nezávislá len \emptyset . Kvôli sporu nech B je taká optimálna nezávislá podmnožina, ktorá neobsahuje x .

- Každý prvok $y \in B$ má váhu najvyššiu $w(x)$, lebo y je nezávislá podmnožina a $w(y) > w(x)$ je spor s voľbou x .
- Dôkaz existencie požadovaného optimálneho riešenia A spravíme jeho konštrukciou.
 - na začiatok položíme $A = \{x\}$; $\{x\}$ je nezávislá podmnožina, preto je A nezávislá.
 - opakovane vyberáme prvok $y \in B - A$ a zachovajúc nezávislosť pridávame ho k A : $A \leftarrow A \cup \{y\}$
 - skončíme, keď $|A| = |B|$. Vtedy $A = B - \{y\} \cup \{x\}$ a podľa váh vieme, že $w(A) = w(B) - w(y) + w(x) \geq w(B)$

□

Lema 2.6 Nech $M = (\mathcal{S}, \mathcal{I})$ je matroid. Ak $x \in \mathcal{S}$ je taký, že nie je rozšírením \emptyset , tak x nie je rozšírením žiadnej nezávislej podmnožiny A .

Dôkaz: Kvôli sporu nech x je rozšírenie A , ale nie rozšírenie \emptyset . Potom $A \cup \{x\}$ je nezávislá podmnožina. Z dedičnosti potom x je tiež nezávislá, čo je spor s predpokladom. □

Lema 2.7 Nech x sme v $GREEDY(M, w)$ vybrali ako prvé. Potom nám ostane *vlastnosť optimálnej štruktúry* doriešiť najťažšiu nezávislú podmnožinu matroidu $M' = (\mathcal{S}', \mathcal{I}')$, kde

$$\left. \begin{array}{l} \mathcal{S}' = \{y \in \mathcal{S} : \{x, y\} \in \mathcal{I}\} \\ \mathcal{I}' = \{B \subseteq \mathcal{S} - \{x\} \mid B \cup \{x\} \in \mathcal{I}\} \end{array} \right\} \text{kontrakcia } M \text{ podľa } x$$

Dôkaz: Ak A je nezávislá podmnožina v M maximálnej váhy, pričom $x \in A$, tak $A' = A - \{x\}$ je nezávislá v M' . Naopak, nezávislá podmnožina A' v M' implikuje nezávislú $A \cup \{x\}$ v M . Keďže $w(A) = w(A') + w(x)$, je $w(A)$ maximálne práve vtedy, keď je maximálne $w(A')$ □

Veta 2.8 Ak $M = (\mathcal{S}, \mathcal{I})$ je vážený matroid s váhovou funkciou w , tak algoritmus $GREEDY(M, w)$ dáva optimálne riešenie.

Dôkaz: Prvky, ktoré nie sú rozšírením \emptyset môžeme zahodiť. Akonáhle máme prvý prvok riešenia, optimálne riešenie pre kontrahovaný matroid M' vedie k optimálnemu riešeniu v M . □