Algoritmi in podatkovne strukture 2

Dinamično programiranje

Luka Fürst

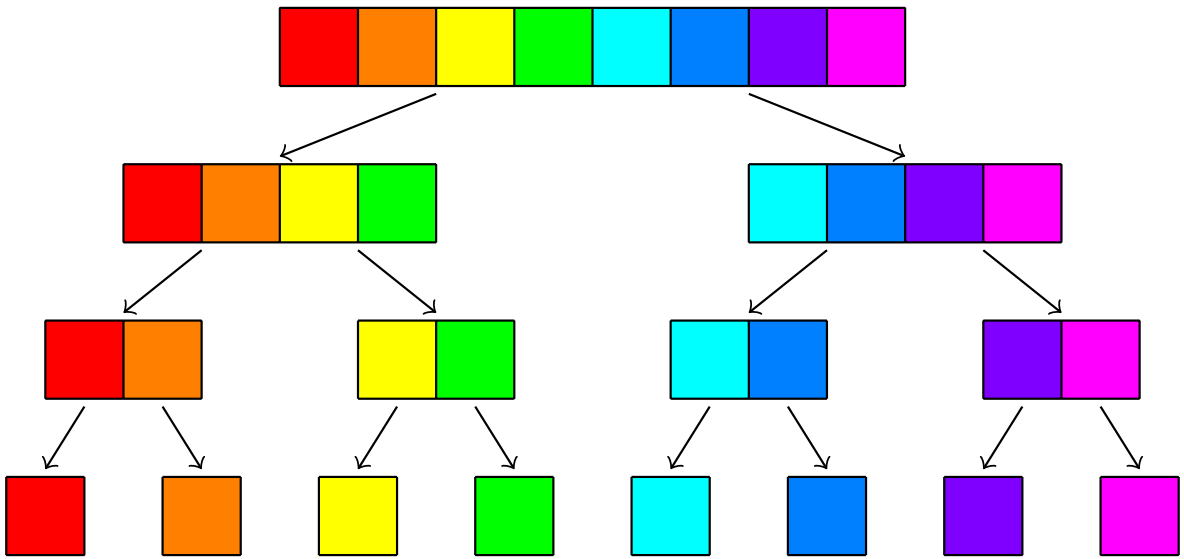
Dinamično programiranje

- Še eden od možnih pristopov k reševanju optimizacijskih problemov (in tudi nekaterih drugih)
- Podoben princip kot pri pristopu deli in vladaj
  - razbij problem na podprobleme enake oblike
- Deli in vladaj
  - podproblemi so med seboj povsem ločeni in jih rešujemo neodvisno drug od drugega
- Dinamično programiranje
  - podproblemi se lahko podvajajo
  - algoritem zasnujemo tako, da se izognemo večkratnemu reševanju istih podproblemov

Deli in vladaj vs. dinamično programiranje

- Deli in vladaj: iskanje minimuma v tabeli

$$m(A,l,r) = \begin{cases} A[l], & \text{če } l = r; \\ \min(m(A,l,\lfloor\frac{l+r}{2}\rfloor), m(A,\lfloor\frac{l+r}{2}\rfloor+1,r)) & \text{sicer} \end{cases}$$

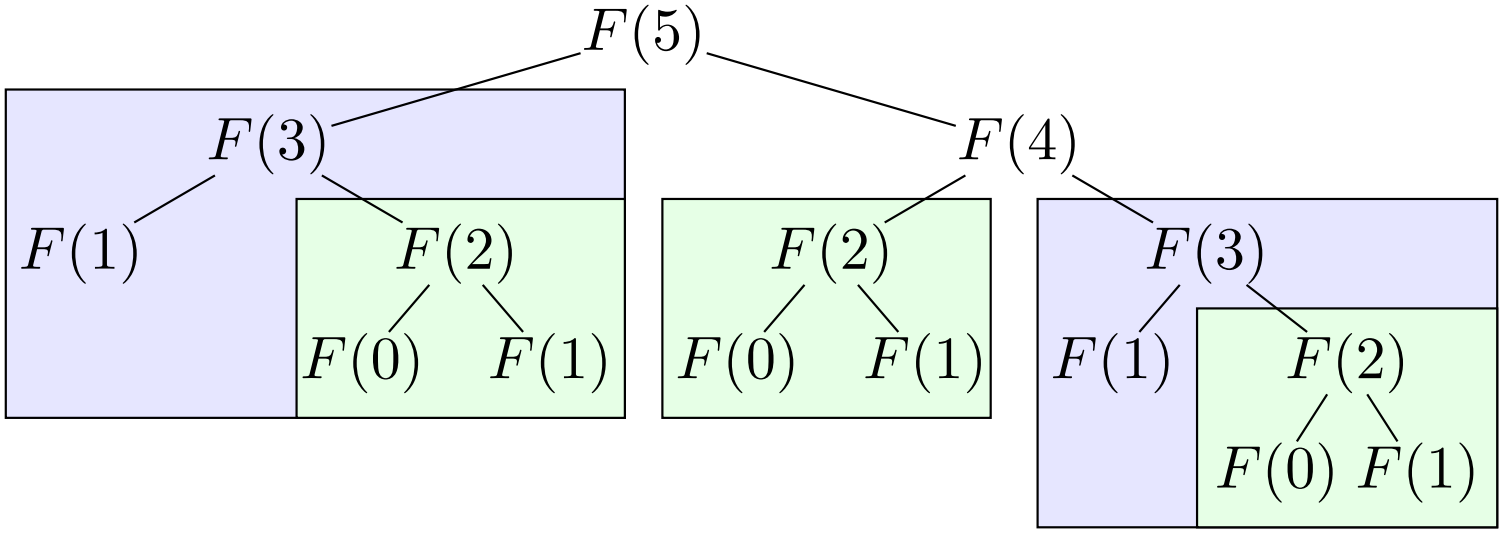


- Podproblemi so med seboj povsem ločeni

Deli in vladaj vs. dinamično programiranje

- Dinamično programiranje: Fibonaccijevo zaporedje

$$F(n) = \begin{cases} n, & \text{če } n \leq 1; \\ F(n-2) + F(n-1) & \text{sicer} \end{cases}$$



- Podproblemi se prekrivajo (in podvajajo)

Dinamično programiranje

- Izhodišče: rekurzivni razcep problema na podprobleme
- Pristop po metodi »od zgoraj navzdol«
  - rekurzija + memoizacija
  - ko podproblem rešimo, si shranimo njegov rezultat
  - pred rekurzivnim klicem preverimo, ali smo podproblem že rešili
  - če smo ga, uporabimo shranjeni rezultat
- Pristop po metodi »od spodaj navzgor«
  - iteracija
  - podprobleme rešujemo v takšnem vrstnem redu, da bomo pri reševanju vsakega podproblema P imeli na voljo rešitve vseh podproblemov, od katerih je P odvisen

0/1-nahrbtnik

Vhod

- prostornina nahrbtnika (V)
- število predmetov (n)
- prostornine predmetov ($v_0, \ldots, v_{n-1}$)
- cene predmetov ($c_0, \ldots, c_{n-1}$)

Izhod

- množica $I \subseteq \{0, \ldots, n-1\}$, tako da je
  - $\sum_{i \in I} v_i \leq V$
  - $C = \sum_{i \in I} c_i$ maksimalna
- včasih nas zanima samo skupna cena predmetov v optimalno napolnjenem nahrbtniku (C)

Primer

Vhod

- $V = 10$
- $n = 4$
- $v_i : 3, 2, 5, 4$
- $c_i : 9, 6, 7, 8$

Izhod

- $I = \{0, 1, 3\}$
- $C = 23$

Rekurzivni razcep

- Polnimo nahrbtnik prostornine V
- Osredotočimo se na prvi predmet (prostornina v_0, cena c_0)
- Možnost 1 (na voljo v vsakem primeru)
  - prvega predmeta ne dodamo v nahrbtnik
  - v tem primeru moramo optimalno napolniti nahrbtnik prostorine V s predmeti 1, 2, ..., $n-1$
- Možnost 2 (na voljo samo v primeru $v_0 \leq V$)
  - prvi predmet dodamo v nahrbtnik
  - v tem primeru moramo optimalno napolniti nahrbtnik prostornine $V - v_0$ s predmeti 1, 2, ..., $n-1$
- Izberemo boljšo od teh dveh možnosti
  - tisto, ki nam dá večjo skupno ceno predmetov v nahrbtniku

Rekurzivni razcep

- Opisana strategija deluje za vsak predmet (ne samo za prvega)
- No, ko nam zmanjka predmetov, je cena optimalnega nahrbtnika enaka 0
- Naj bo $C(i, v)$ cena nahrbtnika prostornine v, ki ga optimalno napolnimo s predmeti i, $i + 1$, $\ldots$, $n - 1$
- Iščemo torej $C(0, V)$

Rekurzivni razcep

- Robni primer

$$C(n, v) = 0 \text{ (za poljuben } v)$$

- Možnost 1 (v vsakem primeru): predmeta i ne dodamo

$$C_{\text{ne}}(i, v) = C(i + 1, v)$$

- Možnost 2 (samo, če je $v_i \leq v$): predmet i dodamo

$$C_{\text{da}}(i, v) = c_i + C(i + 1, v - v_i)$$

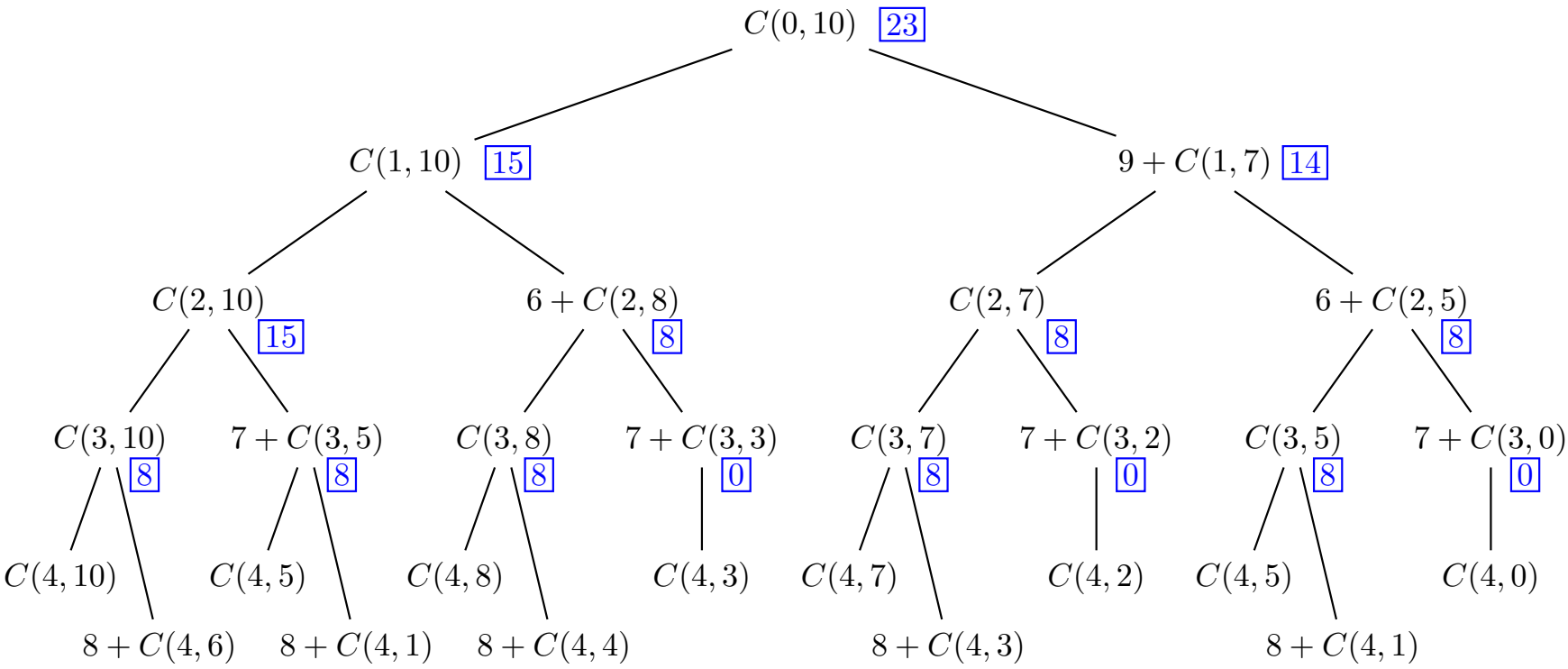
- Izberemo boljšo od obeh možnosti

$$C(i, v) = \begin{cases} C_{\text{ne}}(i, v), & \text{če obstaja samo možnost 1;} \\ \max(C_{\text{ne}}(i, v),\ C_{\text{da}}(i, v)), & \text{če obstajata obe možnosti} \end{cases}$$

Rekurzivni razcep

$$C(i,v) = \begin{cases} 0, & \text{če } i = n; \\ C(i+1,v), & \text{če } i < n \text{ in } v_i > v; \\ \max(C(i+1,v),\ c_i + C(i+1,v-v_i)), & \text{če } i < n \text{ in } v_i \leq v \end{cases}$$

Drevo v našem primeru



Izhodiščna rešitev (od zgoraj navzdol)

- Formulo lahko neposredno prepišemo v rekurzivno funkcijo

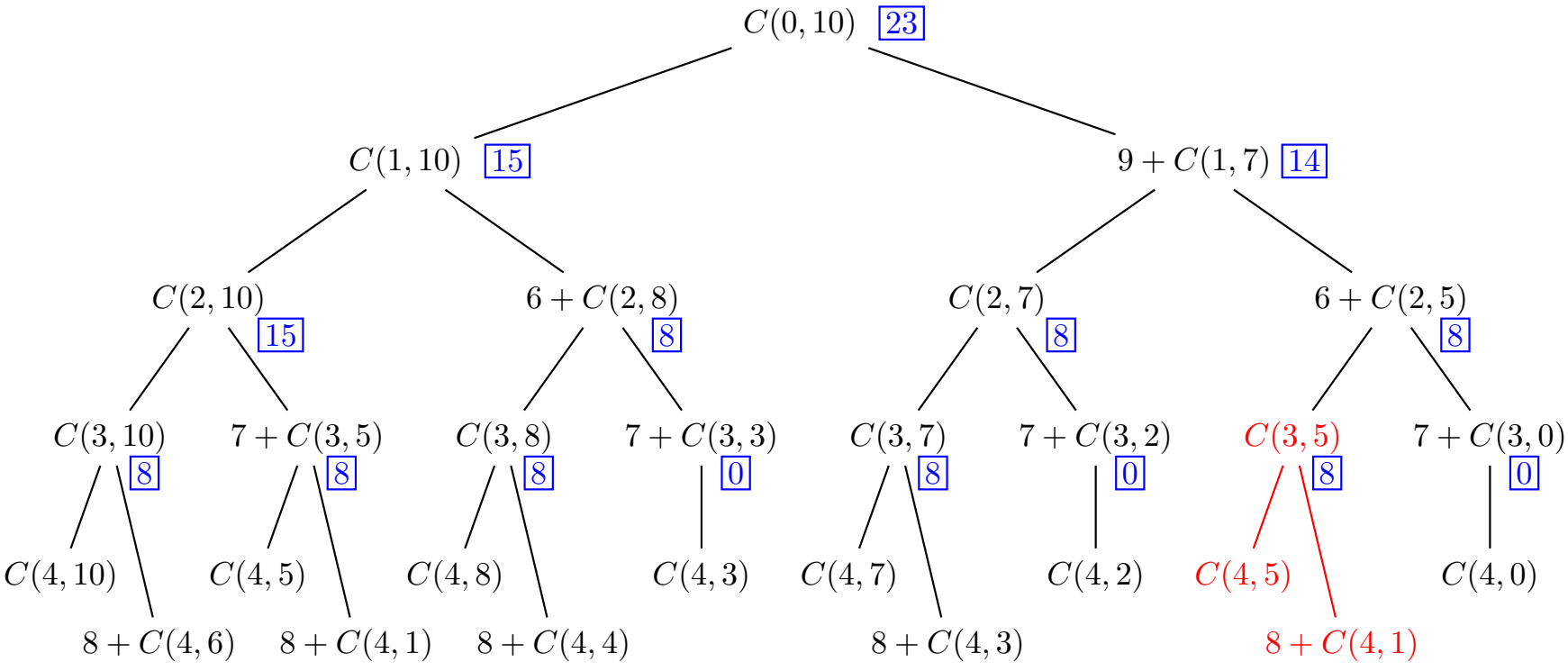
function NAHRBTNIK(V, n, $\langle v_0, \ldots, v_{n-1} \rangle$, $\langle c_0, \ldots, c_{n-1} \rangle$)

    function NAHRBTNIKR(i, v)
        if $i = n$ **then**
            return 0
        $C_{\text{ne}} \leftarrow$ NAHRBTNIKR$(i + 1, v)$
        $C_{\text{da}} \leftarrow C_{\text{ne}}$
        if $v_i \leq v$ **then**
            $C_{\text{da}} \leftarrow c_i +$ NAHRBTNIKR$(i + 1, v - v_i)$
        return $\max(C_{\text{ne}}, C_{\text{da}})$

    return NAHRBTNIKR$(0, V)$

Podvajanje podproblemov

- Ker lahko isto vsoto prostornin pridobimo na več različnih načinov, se bodo nekateri podproblemi podvajali



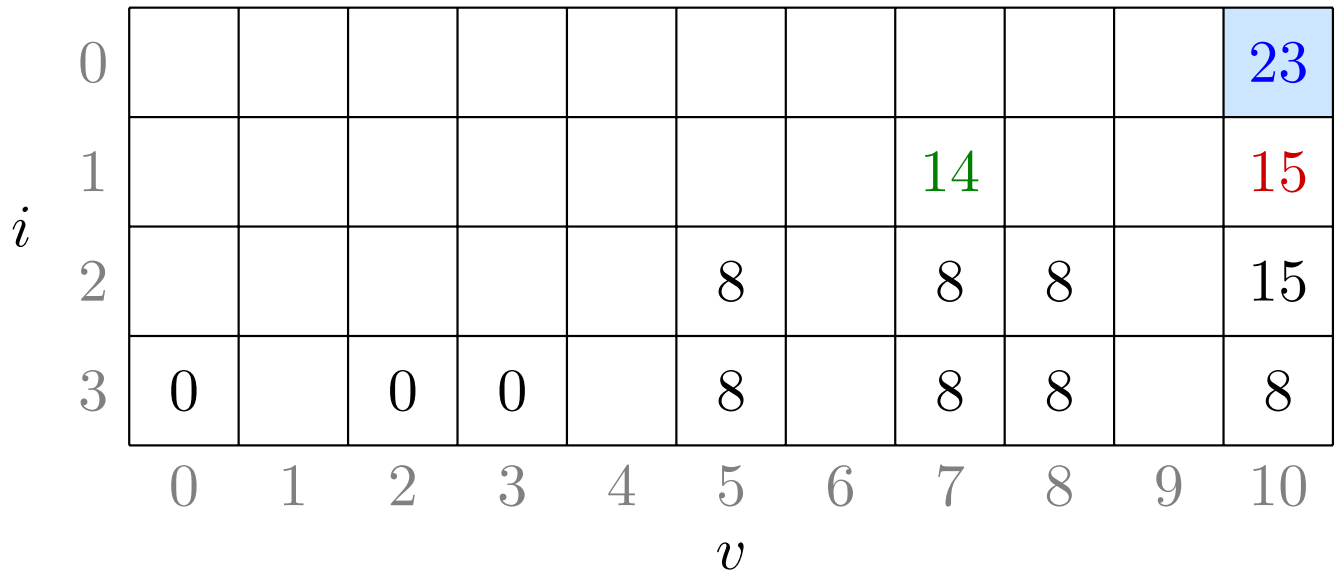
Rešitev z memoizacijo

- Pripravimo si tabelo D za shranjevanje že izračunanih vrednosti funkcije $C(i, v)$
- Tabela je velika $(n + 1) \times (V + 1)$
  - lahko tudi $n \times (V + 1)$, saj so vrednosti $C(n, .)$ enake 0 in jih ni treba pomniti
- $D[i, v]$
  - hrani že izračunano vrednost $C(i, v)$
  - ima vrednost -1, če $C(i, v)$ še nismo izračunali
- Na začetku klica funkcije preverimo $D[i, v]$
  - če je $D[i, v] \geq 0$, vrnemo kar $D[i, v]$
  - sicer izračunamo $C(i, v)$ in rezultat shranimo v $D[i, v]$

Rešitev z memoizacijo

function NAHRBTNIK-MEMO(i, v)
    if $i = n$ **then**
        return 0
    if $D[i, v] \geq 0$ **then**
        return $D[i, v]$
    $C_{\text{ne}} \leftarrow$ NAHRBTNIK-MEMO($i + 1$, v)
    $C_{\text{da}} \leftarrow C_{\text{ne}}$
    if $v_i \leq v$ **then**
        $C_{\text{da}} \leftarrow c_i +$ NAHRBTNIK-MEMO($i + 1$, $v - v_i$)
    $D[i, v] \leftarrow \max(C_{\text{ne}}, C_{\text{da}})$
    return $D[i, v]$

Pomnjenje izračunanih rezultatov



- $C(0, 10) = \max(C(1, 10),\ 9 + C(1, 10 - 3))$
- Ko bomo drugič prispeli do $C(3, 5)$, bo rezultat že izračunan

Računska zahtevnost

- Sprehod po vseh podmnožicah
  - 2^n podmnožic množice predmetov
  - $O(n)$ za vsako
  - skupaj $O(2^n n)$
- Osnovna rekurzivna rešitev
  - drevo ima kvečjemu $2^{n+1} - 1$ vozlišč
  - skupaj $O(2^n)$
- Rešitev z memoizacijo
  - vsak element tabele izračunamo kvečjemu po enkrat
  - $O(nV)$ elementov
  - $O(1)$ za izračun vsakega elementa
  - skupaj $O(nV)$

Mar ni 0/1-nahrbtnik NP-poln?

Mar ni $O(nV)$ polinomska časovna zahtevnost?

Mar ni 0/1-nahrbtnik NP-poln?

Mar ni $O(nV)$ polinomska časovna zahtevnost?

- Ne!
- Za zapis prostornine nahrbtnika zadošča $\lceil \lg V \rceil$ bitov
- Časovna zahtevost je potemtakem $O(2^{b(V)}n)$, kjer je $b(V)$ dvojiška predstavitev prostornine
- Takšni časovni zahtevnosti včasih rečemo psevdopolinomska
- V resnici je seveda eksponentna

Rešitev po metodi »od spodaj navzgor«

- Memoizacijsko tabelo lahko polnimo tudi iterativno
- Za $D[.,.]$ velja ista formula kot za $C(.,.)$

$$D[i,v] = \begin{cases} 0, & \text{če } i = n; \\ D[i+1,v], & \text{če } i < n \text{ in } v_i > v; \\ \max(D[i+1,v],\ c_i + D[i+1,v-v_i]), & \text{če } i < n \text{ in } v_i \leq v \end{cases}$$

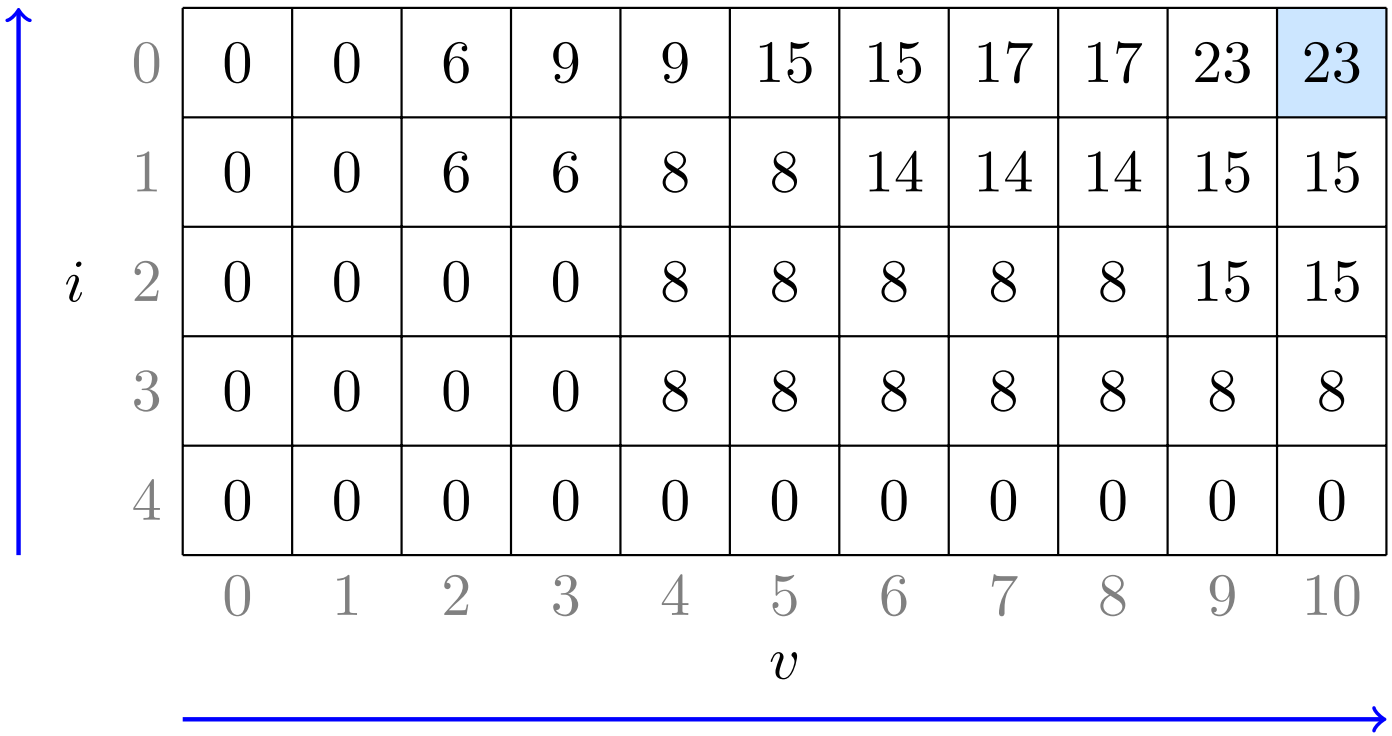
- Vrstico $D[n,:]$ nastavimo na 0
- $D[i,v]$ za $i < n$ računamo po padajočem i in naraščajočem v
- Na ta način bosta ob vsakem izračunu $D[i,v]$ rezultata obeh sestavnih delov ($D[i+1,v]$) in ($D[i+1,v-v_i]$) že znana

Rešitev po metodi »od spodaj navzgor«

function NAHRBTNIK-ITER(V, n, $\langle v_0, \ldots \rangle$, $\langle c_0, \ldots \rangle$)
    izdelaj tabelo D velikosti $(n+1) \times (V+1)$
    $D[n, :] \leftarrow 0$
    for $i \leftarrow n-1$ **downto** 0 **do**
        for $v \leftarrow 0$ **to** V **do**
            $C_{\text{ne}} \leftarrow D[i+1, v]$
            $C_{\text{da}} \leftarrow C_{\text{ne}}$
            if $v_i \leq v$ **then**
                $C_{\text{da}} \leftarrow c_i + D[i+1, v-v_i]$
            $D[i, v] \leftarrow \max(C_{\text{ne}}, C_{\text{da}})$
    return $D[0, V]$

Rešitev po metodi »od spodaj navzgor«

- Napolnimo celotno tabelo
- $O(nV)$



Primerjava obeh pristopov

Rekurzija + memoizacija

- Vrednosti $D[i, v]$ se računajo po potrebi (izračunajo se samo tiste vrednosti, ki prispevajo h končnemu rezultatu)
- Dodatno delo zaradi rekurzije
- Veliko »skakanja« po tabeli (več zgrešitev v predpomnilniku)

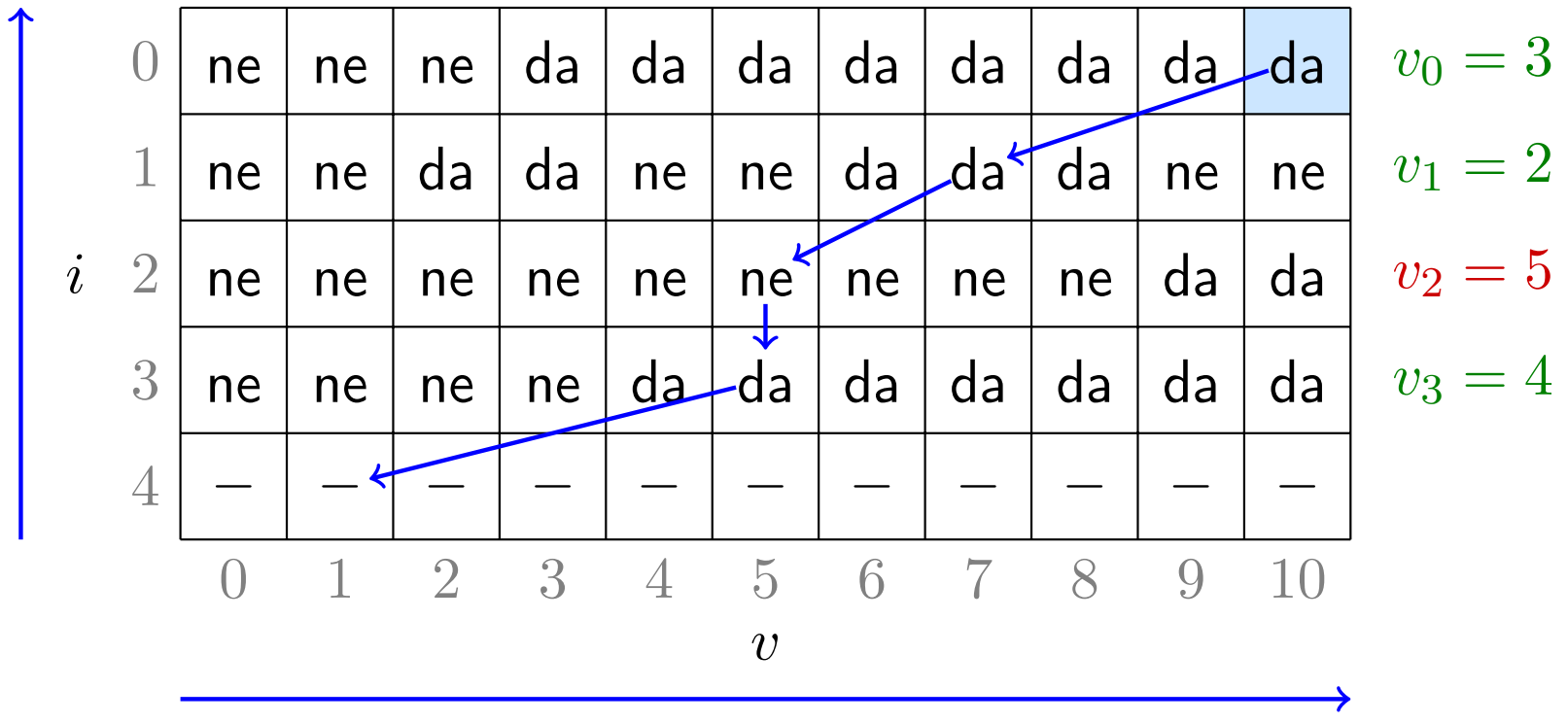
Iteracija

- Izračuna se celotna tabela D (tudi vrednosti, ki ne prispevajo ničesar h končnemu rezultatu)
- Iteracija je cenejša od rekurzije
- Manj »skakanja« po tabeli (manj zgrešitev v predpomnilniku)

Če nas zanimajo tudi izbrani predmeti (ne zgolj skupna cena)

- Poleg tabele D vzdržujemo še enako veliko tabelo odločitev Z
- Če je $D[i, v] = D[i + 1, v]$, nastavimo $Z[i, v] \leftarrow \mathrm{ne}$
  - predmeta i ne vzemi
- Če je $D[i, v] = c_i + D[i + 1, v - v_i]$, nastavimo $Z[i, v] \leftarrow \mathrm{da}$
  - predmet i vzemi
- Pričnemo v celici $Z[0, V]$ in s sledenjem odločitvam rekonstruiramo indekse izbranih predmetov
- Postopek lahko vključimo tako v rekurzivno kot v iterativno rešitev

Če nas zanimajo tudi izbrani predmeti (ne zgolj skupna cena)



Če nas zanimajo tudi izbrani predmeti (ne zgolj skupna cena)

function NAHRBTNIK-PREDMETI(V, n, $\langle v_0, \ldots \rangle$, $\langle c_0, \ldots \rangle$)
    izdelaj tabelo D velikosti $(n+1) \times (V+1)$
    izdelaj tabelo Z velikosti $(n+1) \times (V+1)$
    $D[n, :] \leftarrow 0$
    for $i \leftarrow n-1$ **downto** 0 **do**
        for $v \leftarrow 0$ **to** V **do**
            $C_{\text{ne}} \leftarrow D[i+1, v]$
            $D[i, v] \leftarrow C_{\text{ne}}$
            $Z[i, v] \leftarrow \text{ne}$
            if $v_i \leq v$ **then**
                $C_{\text{da}} \leftarrow c_i + D[i+1, v-v_i]$
                if $C_{\text{da}} > C_{\text{ne}}$ **then**
                    $D[i, v] \leftarrow C_{\text{da}}$
                    $Z[i, v] \leftarrow \text{da}$
    return $\langle D[0, V], \text{NAHRBTNIK-REKONSTRUIRAJ}(Z, V, n) \rangle$

Če nas zanimajo tudi izbrani predmeti (ne zgolj skupna cena)

```
function NAHRBTNIK-REKONSTRUIRAJ(Z, V, n)
    I ← ∅                                   ▷ indeksi izbranih predmetov
    v ← V
    for i ← 0 to n − 1 do
        izbira ← Z[i, v]
        if izbira = da then
            I ← I ∪ {i}
            v ← v − v_i
    return I
```

Prostorsko varčna različica

- Poraba prostora je tudi $O(nV)$
- Če nas zanima samo skupna cena, lahko porabo prostora pri iterativni različici zmanjšamo na $O(V)$
- Izračun celice v vrstici i je odvisen zgolj od celic v vrstici $i + 1$

Prostorsko varčna različica

function NAHRBTNIK-ITER-VARČEN(V, n, ...)
    izdelaj tabeli D in E velikosti $V + 1$
    $D[:] \leftarrow 0$
    for $i \leftarrow n - 1$ **downto** 0 **do**
        for $v \leftarrow 0$ **to** V **do**
            $C_{\text{ne}} \leftarrow D[v]$
            $C_{\text{da}} \leftarrow C_{\text{ne}}$
            if $v_i \leq v$ **then**
                $C_{\text{da}} \leftarrow c_i + D[v - v_i]$
            $E[v] \leftarrow \max(C_{\text{ne}}, C_{\text{da}})$
        $D \leftarrow E$
    return $D[V]$

Dinamično programiranje in optimizacijski problemi

- Optimalna podstruktura
  - optimalna rešitev problema je sestavljena iz optimalnih rešitev podproblemov
- Prekrivajoči se podproblemi
  - isti podproblem se lahko večkrat pojavi, zato shranjujemo rezultate že izračunanih podproblemov ali pa jih rešujemo v takšnem vrstnem redu, da vsakega rešimo le enkrat

Deli in vladaj

- Optimalna podstruktura?
  - DA
- Prekrivajoči se podproblemi?
  - NE, rešujemo jih povsem neodvisno drug od drugega

Požrešna metoda

- Požrešne algoritme običajno implementiramo iterativno (»od spodaj navzgor«), a to ne bi bilo nujno
  - izbiranje intervalov: izberi prvi interval, nato pa rekurzivno izberi še ostale
- Optimalna podstruktura?
  - DA
- Prekrivajoči se podproblemi?
  - NE
- Posebnost
  - pri vsakem rekurzivnem razcepu izvršimo samo požrešno izbiro (npr. izberi nekonflikten interval, ki se prvi konča)
  - ostane nam samo en podproblem (npr. izbira ostalih intervalov)

Najdaljše naraščajoče podzaporedje (NNP)

Vhod

- zaporedje n števil $(a_0, a_1, \ldots, a_{n-1})$

Izhod

- najdaljše zaporedje indeksov $0 \leq i_1 < i_2 < \ldots < i_\ell \leq n-1$, tako da je $a_{i_1} < a_{i_2} < \ldots < a_{i_\ell}$
- včasih nas zanima samo maksimalni ℓ

Primer

Vhod

i	0	1	2	3	4	5	6	7	8	9
a_i	130	50	70	120	20	80	60	110	40	90

Izhod

- $d = 4$
- $I = \langle 1, 2, 5, 7 \rangle$ ali $I = \langle 1, 2, 5, 9 \rangle$

Osnovna ideja

- Naj bo $\ell(i)$ dolžina NNP, ki se prične na indeksu i
- Robni primer
  - $\ell(n) = 0$
- Kako izračunamo $\ell(i)$ za $i < n$?
  - pričnemo na indeksu i (to je prvi člen NNP)
  - obravnavamo vse možne podaljške (vse $j > i$, tako da je $a_j > a_i$)
  - za vsak tak j izračunamo $\ell(j)$
  - izračunamo maksimum vseh dobljenih rezultatov in prištejemo 1

Osnovna ideja

$$\ell(i) = \begin{cases} 0, & \text{če } i = n; \\ 1 + \max\{\ell(j) \mid j > i \wedge a_j > a_i\} & \text{sicer} \end{cases}$$

- »Globalni« ℓ dobimo kot maksimum $\ell(i)$, saj ne vemo vnaprej, kje se NNP prične

$$\ell = \max_{i=0}^{n-1} \ell(i)$$

Primer za vhodno zaporedje $\langle 13, 5, 7, 12 \rangle$

i	0	1	2	3	4	5	6	7	8	9
a_i	130	50	70	120	20	80	60	110	40	90
$\ell(i)$	1	4	3	1	3	2	2	1	2	1

$$\ell(0) = 1 + \max \emptyset$$
$$\ell(1) = 1 + \max\{\ell(2), \ell(3), \ell(5), \ell(6), \ell(7), \ell(9)\}$$
$$\ell(2) = 1 + \max\{\ell(3), \ell(5), \ell(7), \ell(9)\}$$
$$\ldots$$

- Vidimo, da se podproblemi ponavljajo
- $\ell(5)$ izračunamo tako pri $\ell(1)$ kot pri $\ell(2)$

Enostavnejša formulacija

- Na začetek zaporedja dodamo $-\infty$ in tako NNP »prisilimo«, da se začne na indeksu 0

$$\ell(i) = \begin{cases} 0, & \text{če } i = n+1; \\ 1 + \max\{\ell(j) \mid j > i \wedge a_j > a_i\} & \text{sicer} \end{cases}$$

$$\ell = \ell(0) - 1$$

i	0	1	2	3	4	5	6	7	8	9	10
a_i	$-\infty$	130	50	70	120	20	80	60	110	40	90
$\ell(i)$	5	1	4	3	1	3	2	2	1	2	1

Rekurzija z memoizacijo

- Tabela A naj hrani zaporedje $a_0 = -\infty$, a_1, ..., a_n
- Pripravimo si memoizacijsko tabelo D velikosti $n + 2$ in elemente inicializiramo na -1
- Na začetku (takoj za obravnavo robnega primera) preverimo, ali je $D[i] \geq 0$
- Če je, vrnemo kar $D[i]$
- Sicer izračunamo $\ell(i)$ in rezultat shranimo v $D[i]$

Rekurzija z memoizacijo

```
function NNP-MEMO(i)
    if i = n + 1 then
        return 0
    if D[i] ≥ 0 then
        return D[i]
    m ← 0
    for j ← i + 1 to n do
        if A[j] > A[i] then
            m ← max(m, NNP-MEMO(j))
    D[i] ← m + 1
    return D[i]
```

- Funkcijo pokličemo kot $\text{NNP-MEMO}(0)$
- Končni rezultat je $D[0] - 1$

Časovna zahtevnost

- Za vsak $i \in \{0, \ldots, n\}$ izračunamo $\ell(i)$ natanko enkrat
- Izračun $\ell(i)$ za vsak i terja $O(n)$ časa
- Skupaj $O(n^2)$

Iterativna rešitev

- Tabela D hrani vrednosti $\ell(.)$

$$D[i] = \begin{cases} 0, & \text{če } i = n+1; \\ 1 + \max\{D[j] \mid j > i \wedge a_j > a_i\} & \text{sicer} \end{cases}$$

- Če elemente tabele D računamo od desne proti levi, bodo pri računanju $D[i]$ vsi sestavni deli izračuna že na voljo
- Časovna zahtevnost je spet $O(n^2)$

Iterativna rešitev

function NNP-ITER($A = \langle a_0, \ldots, a_{n-1} \rangle$)
    ▷ *Predpostavka:* $A[0] = -\infty$ ◁
    ustvari tabelo D velikosti $n + 2$
    $D[n + 1] \leftarrow 0$
    for $i \leftarrow n$ **downto** 0 **do**
        $m \leftarrow 0$
        for $j \leftarrow i + 1$ **to** n **do**
            if $A[j] > A[i]$ **then**
                $m \leftarrow \max(m, D[j])$
        $D[i] \leftarrow m + 1$
    return $D[0] - 1$

Kako pridobimo NNP (ne pa zgolj ℓ)?

- Vzdržujemo še tabelo Z, ki za vsak i podaja njegovega naslednika v NNP, ki se prične na indeksu i

$$Z[i] = \begin{cases} -1, & \text{če } i = n+1; \\ \arg\max_j \{D[j] \mid j > i \wedge a_j > a_i\} & \text{sicer} \end{cases}$$

- Indekse členov NNP pridobimo tako, da pričnemo z indeksom 0 in sledimo verigi $i \leftarrow Z[i]$

$i \leftarrow 0$
while $i \neq -1$ **do**
    if $i > 0$ **then**
        $\textsc{Print}(i-1)$
    $i \leftarrow Z[i]$

Učinkovitejši pristop

- Bi lahko potovali po zaporedju od leve proti desni in pri tem hranili vsa naraščajoča podzaporedja (NP) do trenutnega indeksa?
- Potem bi seveda zlahka dobili najdaljše NP

Primer

a_i	NP dolžine 1	NP dolžine 2	NP dolžine 3
13	$\langle 13 \rangle$		
5	$\langle 13 \rangle$, $\langle 5 \rangle$		
7	$\langle 13 \rangle$, $\langle 5 \rangle$, $\langle 7 \rangle$	$\langle 5, 7 \rangle$	
12	$\langle 13 \rangle$, $\langle 5 \rangle$, $\langle 7 \rangle$, $\langle 12 \rangle$	$\langle 5, 7 \rangle$, $\langle 5, 12 \rangle$, $\langle 7, 12 \rangle$	$\langle 5, 7, 12 \rangle$

- Na ta način dobimo celo eksponentno časovno zahtevnost, k sreči pa ...

Opažanje 1

- Med vsemi NP iste dolžine lahko ohranimo samo tisto, ki ima najmanjši zadnji člen
- Takšno zaporedje je najobetavnejše
  - ima največ možnosti za rast pri nadaljnjem sprehodu po vhodnem zaporedju
- Zaporedje $\langle 10, 20 \rangle$ je obetavnejše kot $\langle 10, 30 \rangle$
  - če bomo lahko podaljšali zaporedje $\langle 10, 30 \rangle$, bomo lahko podaljšali tudi zaporedje $\langle 10, 20 \rangle$
  - obratno ni nujno res
  - zato ohranimo samo $\langle 10, 20 \rangle$

Opažanje 2

- Zadnji člen NP dolžine k je manjši ali enak predzadnjemu členu NP dolžine $k+1$
- Naj bo Z_k NP dolžine k, Z_{k+1} pa NP dolžine $k+1$
- Prvih k členov zaporedja Z_{k+1} tvori NP dolžine k
- Če bi bil $Z_{k+1}[k-1] < Z_k[k-1]$, bi lahko podzaporedje $Z_{k+1}[0:k-1]$ proglasili za novo NP Z'_k dolžine k, ki bi bilo obetavnejše kot Z_k

Opažanje 3

- Zadnji členi NP naraščajočih dolžin tvorijo strogo naraščajoče zaporedje
- Neposredno sledi iz opažanja 2 in iz dejstva, da so vsa NP strogo naraščajoča

Opažanje 4

- Naj bodo Z_1, Z_2, ..., Z_d NP dolžin 1, 2, ..., d
- Naj bo z_i zadnji člen zaporedja Z_i (za $i \in \{1, \ldots, d\}$)
  - spomnimo se, da velja $z_1 < z_2 < \ldots < z_d$
- Recimo, da smo v vhodnem zaporedju pravkar prebrali element a
- Če je $a > z_d$, potem zaporedje Z_d podaljšamo z elementom a in dobimo NP dolžine $d + 1$
- Zaporedja Z_k dolžine $k < d$ nima smisla podaljšati z elementom a, ker bomo dobili zaporedje dolžine $k + 1$, ki je manj obetavno od obstoječega zaporedja Z_{k+1}

Opažanje 5

- Če je $z_k < a < z_{k+1}$, potem zaporedje Z_k podaljšamo z elementom a in nastalo zaporedje Z'_{k+1} proglasimo za novo NP dolžine $k+1$
- Z'_{k+1} je obetavnejše od Z_{k+1}
- Zaporedja Z_ℓ dolžine $\ell < k$ nima smisla podaljšati z elementom a, saj bi dobili manj obetavno zaporedje dolžine $\ell + 1$ od obstoječega $Z_{\ell+1}$
- Če je $a < z_1$, potem element a proglasimo za novo zaporedje dolžine 1

Posodabljanje NP z elementom a

a_i	NP dolž. 1	NP dolž. 2	NP dolž. 3	NP dolž. 4
5	$\langle 5 \rangle$			
7	$\langle 5 \rangle$	$\langle 5, 7 \rangle$		
8	$\langle 5 \rangle$	$\langle 5, 7 \rangle$	$\langle 5, 7, 8 \rangle$	
2	$\langle 2 \rangle$	$\langle 5, 7 \rangle$	$\langle 5, 7, 8 \rangle$	
3	$\langle 2 \rangle$	$\langle 2, 3 \rangle$	$\langle 5, 7, 8 \rangle$	
4	$\langle 2 \rangle$	$\langle 2, 3 \rangle$	$\langle 2, 3, 4 \rangle$	
6	$\langle 2 \rangle$	$\langle 2, 3 \rangle$	$\langle 2, 3, 4 \rangle$	$\langle 2, 3, 4, 6 \rangle$

Opažanje 6

- Zadošča, da hranimo samo zadnje elemente NP posameznih dolžin
- Odločamo se samo na podlagi zadnjih členov zaporedij
- Kopiranje zaporedja Z_k v prvih k členov zaporedja Z_{k+1} ne vpliva na zadnji element zaporedja Z_{k+1} (ta se vedno nastavi na trenutno obravnavani element)

Iskanje dolžine NNP

function DOLŽINA-NNP($\langle a_0, a_1, \ldots, a_{n-1} \rangle$)
    $d \leftarrow 0$
    for $i \leftarrow 0$ **to** $n - 1$ **do**
        if $d = 0 \vee a_i > z_d$ **then**
            $d \leftarrow d + 1$
            $z_d \leftarrow a_i$
        else
            poišči najmanjši k, tako da je $a_i \leq z_k$
            $z_k \leftarrow a_i$
    return d

Časovna zahtevnost

- Zadnji členi NNP ($z_1, \ldots, z_d$) so naraščajoče urejeni
- Najmanjši k z lastnostjo $a_i \leq z_k$ lahko zato poiščemo z bisekcijo
- $O(n)$ obhodov zanke
- $O(\log n)$ za vsak obhod
- skupaj $O(n \log n)$

Iskanje NNP

- Če želimo poiskati NNP (ne samo njegovo dolžino), vzdržujemo še tabeli I in P
- $I[k]$: indeks zadnjega člena NP dolžine k v vhodnem zaporedju
- $P[i]$: indeks predhodnika člena, ki se v vhodnem zaporedju nahaja na indeksu i
- Zadnji člen NNP je $a_{I[d]}$, nato pa sledimo verigi predhodnikov

Iskanje NNP

i	a_i	z	I	$P[i]$
0	130	$\langle 130 \rangle$	$\langle 0 \rangle$	–
1	50	$\langle 50 \rangle$	$\langle 1 \rangle$	–
2	70	$\langle 50, 70 \rangle$	$\langle 1, 2 \rangle$	1
3	120	$\langle 50, 70, 120 \rangle$	$\langle 1, 2, 3 \rangle$	2
4	20	$\langle 20, 70, 120 \rangle$	$\langle 4, 2, 3 \rangle$	–
5	80	$\langle 20, 70, 80 \rangle$	$\langle 4, 2, 5 \rangle$	2
6	60	$\langle 20, 60, 80 \rangle$	$\langle 4, 6, 5 \rangle$	4
7	110	$\langle 20, 60, 80, 110 \rangle$	$\langle 4, 6, 5, 7 \rangle$	5
8	40	$\langle 20, 40, 80, 110 \rangle$	$\langle 4, 8, 5, 7 \rangle$	4
9	90	$\langle 20, 40, 80, 90 \rangle$	$\langle 4, 8, 5, 9 \rangle$	5