Dinamično programiranje

Problemi, ki jih rešujemo z dinamičnim programiranjem

- Problem, ki ga rešujemo z rekurzivnim razbitjem na podprobleme
  - podproblemi so enake oblike kot osnovni problem, le manjši
- Rešitev problema sestavimo iz rešitev posameznih podproblemov
  - rekurzivna funkcija
- Podproblemi se med seboj prekrivajo

Dinamično programiranje

- Možnost 1
  - ohranimo rekurzivno zgradbo (»od zgoraj navzdol«), vendar pa pomnimo vrednosti, ki jih že izračunamo
  - memoizacija (ne memo-R-izacija!)
- Možnost 2
  - problem rešujemo iterativno (»od spodaj navzgor«)
  - najprej rešimo najmanjše podprobleme, potem pa njihove rešitve postopoma sestavljamo v rešitve večjih podproblemov

Fibonaccijevo zaporedje

- Naloga
  - Fibonaccijevo zaporedje se prične s členoma $a_0 = 0$ in $a_1 = 1$, vsak naslednji člen pa se izračuna po formuli $a_i = a_{i-1} + a_{i-2}$. Poišči člen a_n pri podanem n.
- Vhod
  - celo število $n \in [1, 90]$
- Izhod
  - Fibonaccijevo število a_n ($< 2^{63}$)
- Primer vhoda in izhoda

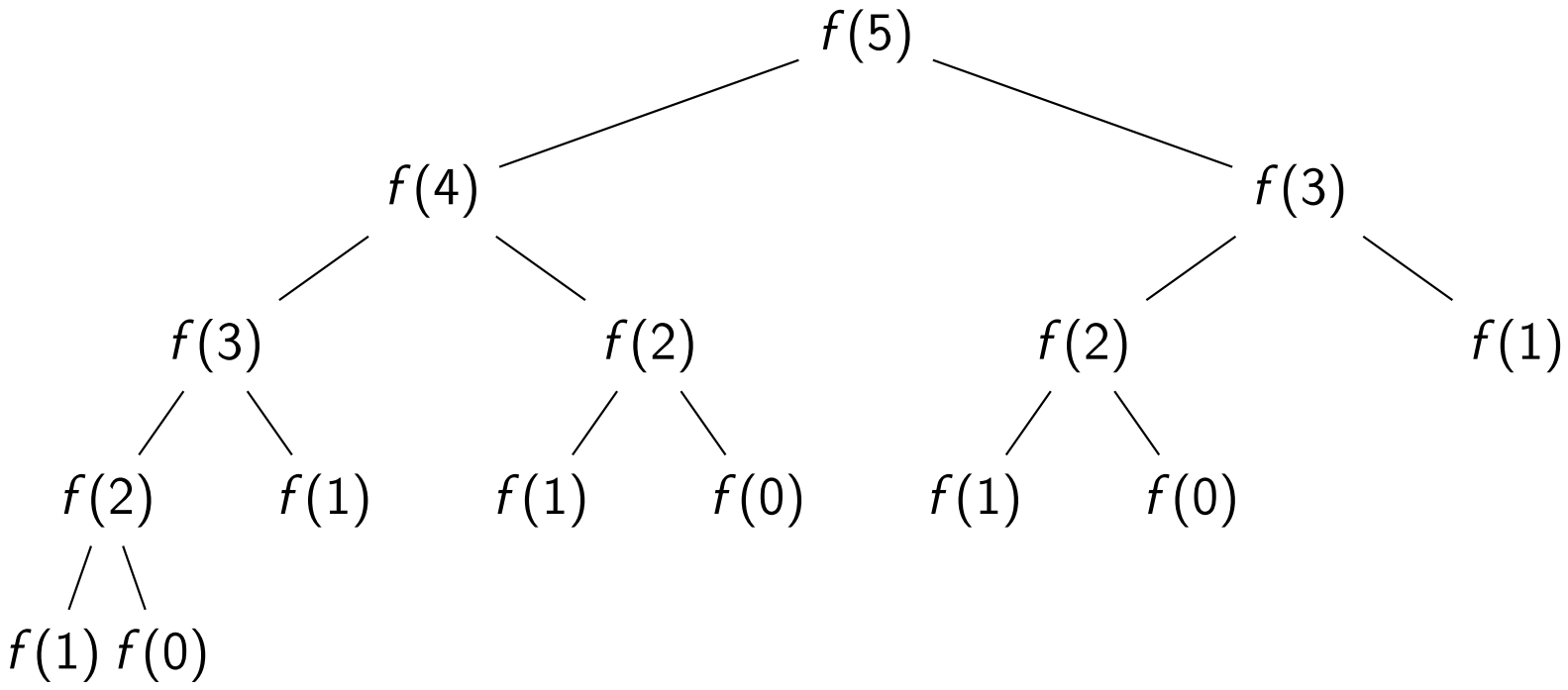
7
13

Osnovna rekurzivna rešitev

```
long fib(int n) {
    if (n < 2) {
        return n;
    }
    return (fib(n - 1) + fib(n - 2));
}
```

Osnovna rekurzivna rešitev

- Podproblemi se med seboj prekrivajo



Rešitev z memoizacijo

- Že izračunane vrednosti shranjujemo v tabeli `M`
- Preden pričnemo z računanjem vrednosti za `n`, pogledamo v tabelo `M`, ali smo jo že izračunali
  - `M[n] == 0` pomeni, da je še nismo
  - `M[n] > 0` pomeni, da smo jo že (vrednost je `M[n]`)

```
long M[91];

long fib(int n) {
    if (n < 2) {
        return n;
    }
    if (M[n] > 0) {
        return M[n];
    }
    M[n] = fib(n - 1) + fib(n - 2);
    return M[n];
}
```

Iterativna rešitev

- Tabelo `M` gradimo po naraščajočih indeksih

```
long M[91];

long fib(int n) {
    M[0] = 0;
    M[1] = 1;
    for (int i = 2;  i <= n;  i++) {
        M[i] = M[i - 1] + M[i - 2];
    }
    return M[n];
}
```

Iterativna rešitev (brez tabele)

```
long fib(int n) {
    if (n < 2) {
        return n;
    }
    long prejsnji = 0;
    long trenutni = 1;
    for (int i = 2;  i <= n;  i++) {
        long predprejsnji = prejsnji;
        prejsnji = trenutni;
        trenutni = predprejsnji + prejsnji;
    }
    return trenutni;
}
```

Problem nahrbtnika

- Naloga
  - Podanih je n predmetov s prostorninami $v_0, \ldots, v_{n-1}$ in cenami $c_0, \ldots, c_{n-1}$. Kolikšna je največja skupna cena predmetov, ki jih lahko spravimo v nahrbtnik prostornine V?
- Vhod
  - celo število $V \in [0, 1000]$
  - celo število $n \in [1, 1000]$
  - cela števila $v_i \in [1, 1000]$
  - cela števila $c_i \in [1, 1000]$
- Izhod
  - skupna cena izbranih predmetov

Problem nahrbtnika

- Primer vhoda in izhoda

10 5 3 4 7 2 3 5 6 8 9 1
20

Izberemo prvi, drugi in četrti predmet.

Osnovna rekurzivna rešitev

- $C(i, V)$: maksimalna cena nahrbtnika prostornine V, pri čemer lahko v nahrbtnik dodajamo predmete od i-tega naprej
- Zanima nas $C(0, V)$
- Kako izračunamo $C(i, V)$?
  - Možnost 1
    - predmeta i ne dodamo v nahrbtnik
    - cena nahrbtnika: $C(i + 1,\ V)$
  - Možnost 2
    - predmet i dodamo v nahrbtnik (če je dovolj prostora)
    - cena nahrbtnika: $c_i + C(i + 1,\ V - v_i)$
  - Izberemo boljšo od obeh možnosti
    - $C(i, v) = \max\{c_i + C(i + 1, V - v_i),\ C(i + 1, V)\}$

Osnovna rekurzivna rešitev

```
int poisci(int n, int ix, int* cene, int* prostornine, int mejaV) {
    if (ix == n) {
        return 0;
    }

    // predmeta z indeksom ix ne dodamo v nahrbtnik
    int najCena = poisci(n, ix + 1, cene, prostornine, mejaV);

    if (prostornine[ix] <= mejaV) {
        // predmet z indeksom ix dodamo v nahrbtnik
        int c = poisci(n, ix + 1, cene, prostornine,
                       mejaV - prostornine[ix]);

        // boljša od obeh možnosti
        najCena = MAX(najCena, c + cene[ix]);
    }
    return najCena;
}
```

Rešitev z memoizacijo

- `M[i][V]`: že izračunana vrednost $C(i, V)$

```
int M[1001][1001];

int poisci(int n, int ix, int* cene, int* prostornine, int mejaV) {
    if (ix == n) { return 0; }
    if (M[ix][mejaV] > 0) { return M[ix][mejaV]; }  // preverimo

    int najCena = poisci(n, ix + 1, cene, prostornine, mejaV);
    if (prostornine[ix] <= mejaV) {
        int c = poisci(n, ix + 1, cene, prostornine,
                       mejaV - prostornine[ix]);
        najCena = MAX(najCena, c + cene[ix]);
    }
    M[ix][mejaV] = najCena;   // pomnimo
    return najCena;
}
```

Iterativna rešitev

- vrednosti `M[i][V]` računamo po padajočih vrednostih `i` in naraščajočih vrednostih `V`

```
int M[1001][1001];
int poisci(int n, int* cene, int* prostornine, int mejaV) {
    for (int i = n - 1;  i >= 0;  i--) {
        int cena = cene[i];
        int prostornina = prostornine[i];
        for (int V = 1;  V <= mejaV;  V++) {
            if (prostornina > V) {
                M[i][V] = M[i + 1][V];
            } else {
                M[i][V] = MAX(M[i + 1][V],
                              cena + M[i + 1][V - prostornina]);
            }
        }
    }
    return M[0][mejaV];
}
```

Naloga za vas

- Dopolnite iterativno različico funkcije `poisci` tako, da bo izpisala predmete, ki sodijo v optimalni nahrbtnik

Najdaljše naraščajoče podzaporedje

- Naloga
  - Za podano zaporedje n števil poišči dolžino najdaljšega naraščajočega podzaporedja. Podzaporedje ni nujno strnjeno.
- Vhod
  - celo število $n \in [1, 1000]$
  - cela števila $z_0, \ldots, z_{n-1} \in [-10^9, 10^9]$
- Izhod
  - dolžina najdaljšega naraščajočega podzaporedja
- Primer vhoda in izhoda

```
10
13 5 7 12 1 8 6 11 4 9
```

```
4
```

Eno od optimalnih podzaporedij je 5, 7, 8, 11.

Osnovna rekurzivna rešitev

- $D(k)$: dolžina najdaljšega naraščajočega podzaporedja, ki se prične s členom z_k
- Zanima nas $\max\{D(0), D(1), \ldots, D(n-1)\}$
- $D(n) = 0$
- $D(k) = \max\{1, \max\{D(i) + 1 \mid i > k \text{ in } z_i > z_k\}\}$

Osnovna rekurzivna rešitev

```
int dolzina(int n, int k, int* zaporedje) {
    if (k == n) { return 0; }
    int najD = 1;
    for (int i = k + 1;  i < n;  i++) {
        if (zaporedje[i] > zaporedje[k]) {
            najD = MAX(najD, dolzina(n, i, zaporedje) + 1);
        }
    }
    return najD;
}
int main() {
    ...
    int najDolzina = 0;
    for (int k = 0;  k < n;  k++) {
        najDolzina = MAX(najDolzina, dolzina(n, k, zaporedje));
    }
    printf("%d\n", najDolzina);
    ...
}
```

Rešitev z memoizacijo

```
int M[1001];

int dolzina(int n, int k, int* zaporedje) {
    if (k == n) {
        return 0;
    }
    if (M[k] > 0) {
        return M[k];
    }

    int najD = 1;
    for (int i = k + 1;  i < n;  i++) {
        if (zaporedje[i] > zaporedje[k]) {
            najD = MAX(najD, dolzina(n, i, zaporedje) + 1);
        }
    }
    M[k] = najD;
    return najD;
}
```

Politična nasprotja

- Naloga
  - Na $l + d + c$ sedežev v ravni vrsti želimo razporediti l levičarjev, d desničarjev in c centristov, tako da levičar in desničar nikoli ne bosta sedela skupaj. Na koliko načinov lahko to naredimo?
- Vhod
  - cela števila $l \in [0, 20]$, $d \in [0, 20]$, $c \in [0, 20]$
- Izhod
  - število načinov ($< 2^{63}$)

Politična nasprotja

- Primer vhoda in izhoda

2 3 2
15

Možne razporeditve so LLCDDDC, LLCDDCD, LLCDCDD, LLCCDDD, LCLCDDD, LCDDDCL, DDDCLLC, DDDCLCL, DDDCCLL, DDCLLCD, DDCDCLL, DCLLCDD, DCDDCLL, CLLCDDD in CDDDCLL.