Jméno: Petr Valenta **UČO:** 487561

2. [20 bodů] V Algoritmické klubovně se každý rok organizuje turnaj, kterého se účastní n hráčů rozesazených v pořadí $1, 2, \ldots, n$ kolem kruhového stolu. Úkolem hráčů je vytvořit dvojice a položit na stůl provázek od jednoho hráče dvojice k druhému. Provázky se nesmí křížit. Každý provázek má svou cenu. Cílem hry je položit na stůl takové provázky (vytvořit takové dvojice), které budou mít v součtu maximální možnou cenu.

Vstupem problému Algoritmické klubovny je dvourozměrná matice $M[1 \ldots n, 1 \ldots n]$ přirozených čísel, kde $M[i, j] = M[j, i]$ je cena provázku, který spojuje hráče i a j. Pravidla požadují, aby současně s provázkem spojujícím hráče i a j ($i < j$) nebyl použit žádný provázek spojující hráče x a y pro $i < x < j$ a pro $y < i$ anebo $y > j$. Použitím principů dynamického programování navrhněte algoritmus polynomiální časové složitosti, který vybere provázky splňující pravidla tak, aby součet jejich cen byl maximální možný.

Příklad. Jestliže se turnaje účastní 8 hráčů a hráči 2 a 6 vytvoří dvojici, tak hráči 7, 8 a 1 nemohou vytvořit dvojici se žádným z hráčů 3, 4 a 5. Jestliže vytvoří dvojici hráči 1 a 7, tak hráč 8 zůstane sám.

Rozdělení na podproblémy

Problém definuji jako uspořádanou k-tici, $k\epsilon[0, n]$, ve které se nachází prvky reprezentující hráče. Prvky jsou seřazeny vzestupně a na sousedících pozicích musí být prvky reprezentující sousedící hráče.

Taková uspořádaná k-tice může tedy být například: $()$, (3), $(4, 5, 6)$ nebo $(n-3, n-2, n-1, n)$. k-tici splňující tyto předpoklady budu označovat t_k. $t_k[x]$ označuje x-tý prvek takové k-tice. Množinu všech možných k-tic které je možné vytvořit ze vstupní posloupnosti, budu zapisovat jako T_k.

Problém dělím na podproblémy tak, že volím, s kým se spojí hráč na první pozici v k-tici reprezentující daný problém. Hráč se může spojit s kterýmkoliv hráčem v k-tici. I sám se sebou — tato volba má nulovou cenu a znamená, že se hráč nespojí s žádným jiným hráčem.

V každé z k voleb se poté k-tice dělí na vnitřní i-tice a vnější o-tice dle zvoleného spojení. Například pro šestici $(1, 2, 3, 4, 5, 6)$ a volbu $(1, 3)$ je vnitřní jednotice (2) a vnější trojice $(4, 5, 6)$. Velikost i-tice a o-tice je možné vyjádřit tímto vztahem.

$$i + o = \begin{cases} k-1, & \text{pro volbu} \quad \text{(x,x)} \\ k-2, & \text{pro volbu} \quad \text{(x,y), y} \neq \text{x} \end{cases}$$

Problém má k-voleb, každá z nich má 2 podproblémy s průměrnou velikostí $(k-2)/2$.

Tyto vnitřní i-tice a vnější o-tice jsou podproblémy. Výsledná cena problému je maximum ze součtů ceny volby a cen podproblémů volby.

Rekurentní rovnice

$$OPT(t_k) = \begin{cases} 0, & k = 0 \\ 0, & k = 1 \\ M[t_k[1], t_k[2]], & k = 2 \\ \max\limits_{1 \leq x \leq k} (M[t_k[1], t_k[x]] + OPT(t_i) + OPT(t_o)), & k > 2 \end{cases}$$

t_i je vnější i-tice, její velikost je $i = x - 2$. Prvky $a\epsilon t_i$ nabývají hodnot $t_k[1] < a < t_k[x]$.

t_o je vnější o-tice, její velikost je $o = k - x$. Prvky $b\epsilon t_o$ nabývají hodnot $t_k[x] < b \leq t_k[k]$.

Pořadí podproblémů Podproblémy t_k , $k\epsilon[2, n]$ budeme řešit vzestupně. Na pořadí podproblémů stejné velikosti nezáleží.

Jméno: Petr Valenta **UČO:** 487561

Algoritmus

```
function play(M)
    n ← number of rows of M
    values[t_0], choices[t_0] ← 0
    for all t_1 in T_1 do
        values[t_1], choices[t_1] ← 0
    for all t_2 in T_2 do
        values[t_2] ← M[t_2[1], t_2[2]]
        choices[t_2] ← t_2[2]
    for k in [ 3 , n ] do
        for all t_k in T_k do
            maximumValue ← 0
            choice ← 0
            for all elements x in t_k do
                t_i ← elements of t_k between first and x
                t_o ← elements of t_k between x and last
                value ← M[t_k[1] , x] + values[inner] + values[outer]
                if value > maximumValue then
                    maximumValue ← value
                    choice ← x
            values[t_k] ← maximumValue
            choices[t_k] ← choice
    solution(values, choices, t_n)
```

Sestavení řešení U každého vyřešeného podproblému t_k si kromě hodnoty jeho řešení uložím navíc i číslo hráče, který byl spojen s hráčem $t_k[1]$. K získání řešení procházím po problémech shora dolů — začnu problémem $t_{k=n}$, poznamenám si zvolené spojení hráčů, rozdělím problém na podproblémy t_i a t_o a rekurzivně opakuji.

```
function solution(values, choices, t_k)
    if k ≥ 2 then
        output ← ( t_k[1] , choices[t_k] )
        t_i ← elements of t_k between first and choices[t_k]
        t_o ← elements of t_k between choices[t_k] and last
        solution(values, choices, t_i)
        solution(values, choices, t_o)
```

Zdůvodnění korektnosti

Optimální strukturou řešení je posloupnost dvojic. Každá dvojice reprezentuje pár vytvořený dvěmi hráči. Jestliže hráč žádný pár nevytvořil, je v páru sám se sebou. Pro zjednodušení důkazu tuto posloupnost seřadíme vzestupně podle prvního hráče ve dvojici. Maximální délka této posloupnosti je n.

Jestliže hledám posloupnost dvojic takovou, že součet cen provázků bude co největší, pak skutečně optimální řešení $SOPT$ tuto posloupnost najde.

Předpokládejme, že do jisté dvojice se $SOPT$ a OPT rozhodovaly stejně. V této volbě se poprvé neshodly. Protože je posloupnost seřazená, první hráč z dvojice je daný. OPT vybere největší hodnotu z optimálních řešení všech podproblémů, což znamená, že $SOPT$ si zvolilo stejně drahý, ale jiný podproblém, nebo dražší podproblém. V případě, že si $SOPT$ zvolil dražší podproblém, pak je striktně horší. V případě, že si zvolil jiný, stejně ohodnocený podproblém, na zálkadě exchange argumentu tvrdím, že si mohl zvolit stejný podproblém jako OPT a výsledek by to nezměnilo.

Protože sestavujeme řešení zdola nahoru, a při řešení problému se odkazujeme na menší podproblémy, je zaručeno, že optimální řešení podproblémů bude v čase počítání problému známo.

Analýza složitosti

Algoritmus se sestává z množin podproblémů T_0 , ... , T_n. Množina T_0 obsahuje vždy pouze jeden podproblém, v analýze ji tedy můžeme zanedbat. Velikost množiny T_k je $n-k$. V každém podproblému t_k z množiny T_k pro $k > 2$ se vykoná $6+3(k-3)$ dotazů. V množině T_2 provede každý podproblém pouze jeden dotaz. Celkový počet dotazů je tedy:

$$1 \times (n-2) + \sum_{k=3}^{n} (n-k)(n+3(k-3)) = \frac{n^3}{2} - \frac{3n^2}{2} - n + 4$$

Složitost sestavení řešení je zanedbatelná. Celková složitost je $\mathcal{O}(n^3)$ — kubická.

Algoritmus si ukládá pole, které pro každý podproblém obsahuje hodnotu jeho optimálního řešení a volbu, pomocí které bylo tohoto řešení dosaženo. Velikost těchto informací nezávisí na velikosti vstupu, pojmenujeme si je tedy C. Velikost pole je tedy:

$$\sum_{k=1}^{n} (n-k) \times C = (\frac{n^2}{2} - \frac{n}{2}) \times C$$

Prostorová náročnost je $\mathcal{O}(n^2)$ — kvadratická.