1.

Dôkaz, že LP je NP-úplné:

1. Dokázať, že $LP \in NP$
2. Dokázať, že LP je NP-ťažký

1. Dokážem to tým že vytvorím nedeterministický viac-páskový TS s označením M, ktorý rozhoduje LP v polynomiálnom čase. Chovanie M je následovné:

Na prvej páske má M svoj vstup vo formáte:
$\Delta \langle H \rangle \# \langle M \rangle \# \langle N \rangle \# \langle S \rangle \# \langle U \rangle \Delta \Delta^{\omega}$, pričom $\langle H \rangle$ je kód množiny hostí, $\langle M \rangle$ je zakódovaná tabuľka relácie M, $\langle N \rangle$ je zakódovaná tabuľka relácie N, $\langle S \rangle$ je počet stolov S a $\langle U \rangle$ je počet miest u stola U. Kontrola vstupu má polynomiálnu zložitosť. Ak je vstup validný tak pokračuje inak odmieta. M nedeterministicky vyberie maximálne V hostí a presunie ich na druhú pásku. Výber a presun má maximálne polynomiálnu zložitosť. Následne skontroluje či hostia na druhej páske nemajú medzi sebou reláciu N. Ak nie tak pokračuje, ak áno tak odmieta. Táto kontrola prebehne v polynomiálnom čase. Potom kontroluje či hostia na druhej páske niesú v relácii M s hosťami na 1. páske. Ak sú tak odmieta a niesú tak pokračuje. Táto kontrola prebehne v polynomiálnom čase. Ak vybraný hostia prejdu všetkými kontrolami tak ich M zakóduje ako množinu hostí pri stole a vloží na výstupnú

pásku. Následne ich vymaže z 2. pásky. Ak ešte ostávajú hostia z množiny H, tak tento proces opakuje, pokiaľ bude mať ešte voľných hostí alebo počet stolov. Zaradenie hostí ku stolom týmto spôsobom bude stále v polynomiálnom čase. Ak mu už nezostanú voľný hostia na výber do 2. pásky, tak vracia výsledok na výstupnej páske. Ak zaplní všetky stoly a stále mu ostávajú voľný hostia tak odmieta.

Z tohto vyplíva že získanie riešenia na tomto TS bude najhorším $O(n^k)$ a teda $LP \in NP$.

(2.) Dôkaz, že LP je NP-ťažký sa dá spraviť pomocou polynomiálnej redukcie z NP-problému na LP. Ja som si zvolil k-coloring graph problém. Redukčná funkcia $\varphi: (0,1,\#)^* \to (0,1,\#)^*$ sa správa takto:

Ak dostane na vstup nevalidnú inštanciu k-coloring graph problému, tak vracia inštanciu LP problému kde $H=\{1,2,3\}$, $N=\emptyset$, $M=\emptyset$, $S=0$, $V=0$. Pre zjednodušenie túto inštanciu pomenujem LP_false. Ak dostane správnu inštanciu tak nad ňou spustí simuláciu k-coloring problému. Keď sa simulácia skončí zlyhaním tak funkcia vracia LP_false. Ak simulácia nájde riešenie tak funkcia vráti inštanciu LP, kde H bude množina vrcholov v grafe, M bude prázdna množina, relácia N bude platiť pre všetky vrcholy, ktoré sú spojené hranou v grafe, počet stolov bude počet zafarbení vrcholov grafu a počet miest pri stole bude maximálny počet vrcholov zafarbených jednou farbou. Takto vieme zaistiť že ak bude existovať riešenie k-coloring problému, tak ho rovnako vieme nájsť aj vo výstupe funkcie φ pre problém LP, čím uspešne redukujem NP-úplný problém k-coloring graph na LP problém v polynomiálnom čase a tým pádom dokázať že LP je NP-ťažký.

Keďže sa mi podarilo dokázať že LP problém $\in$ NP a je NP-ťažký, tak môžem prehlásiť že LP je aj NP-úplný problém.

2.

```
vloz_a_urez(k){                                  1
    node = get_tail();
    while True {
        if (list_empty()){                       1
            new_list(k);                         1
            break;}                              1
        if (k > node.value){                     1
            new_node(node.forw, k);              1
            break;}                              1
        if (node == get_front()){                1
            new_node(node.back, k);              1
            break;}                              1
        node = node.back                         1
        print_and_delete(node.front)             1
    }
}
```

while = O(n)

Najhoršia zložitosť funkcie: O(n)

Pomocné funkcie a premenné

node → štruktúra položky v zozname

skladá sa z 3 premenných:

value : hodnota položky

forw : ukazateľ na pol. vpredu

back : ukazateľ na pol. vzadu

get_tail(), get_front() – vráti ukazateľ na začiatok / koniec zoznamu, O(1)

new_node(pos, value) – vytvorí novú položku v zozname s hodnotou value a vloží ju na pozíciu pos, O(1)

print_and_delete(pos) – vytlačí hodnotu položky na pozícii pos a následne ju zmaže, O(1)

new_list(value) – do prázdneho listu vloží položku s hodnotou value, O(1)

list_empty() – kontrola prázdnosti listu, O(1)

Amortizovaná zložitosť pre n volaní funkcie nad novo iniciatizovaným prázdnym listom:

Zložitosť n volaní funkcie vloz_a_uprav rozdelím do 2 častí:

Časť vlož: n-krát vložím položku do zoznamu. Čiže zložitosť tejto časti bude n

Časť uřez: Aby som mohol odstrániť položky zo zoznamu, musel som ich tam najprv pridať. Čiže v najhoršom ak pridám n položiek v časti ulož, tak v časti uřez môžem odstrániť maximálne n-1 položiek. Teda časť uřez bude mať zložitosť maximálne n-1.

Celková amortizovaná zložitosť n volaní funkcie ulož_a_uřez bude teda $O(2n-1)$, čo je teda $O(n)$