Analýza a zložitosť algoritmov

Praktická implementácia a analýza algoritmu

Autor: Marek Čederle
AIS ID: 121193 (xcederlem)
Cvičiaci: Ing. Ján Bartoš
(Utorok 10:00)

Úloha 1

Algoritmus na plánovanie s termínmi (scheduling with deadlines algorithm) funguje nasledovne:

Máme vstupný neusporiadaný vektor prác (jobov), kde každá má pridelené svoje ID číslo, deadline a jej profit. Následne zavoláme funkciu na plánovanie. Funkcia najprv tento neusporiadaný vektor usporiada podľa profitov zostupne (to znamená že prvý job bude s najvyšším profitom, druhý s nižším atď.). Na usporiadanie som použil Quick Sort algoritmus. Quick sort funguje na princípe divide and conquer (rozdeľuj a panuj). Zvolíme si pivot, ktorý dáme na jeho finálne miesto kde sa bude nachádzať vo výslednom poli a následne rekurzívne voláme quick sort, kde pošleme ľavú časť poľa od pivotu a pravú časť poľa od pivotu. Jeho najhoršia časová zložitosť (Worst case time complexity) je síce Big O(n^2) avšak pre malé datasety, ktoré sú vo forme poľa má priemernú (aj najlepšiu) časovú zložitosť Big O(n * log(n)). Worst case môže nastať pri zlom zvolení pivotu. Ak však pivot zvolíme tak že je približne v polovici výsledného poľa, tak nám to skoro zaručuje jeho rýchlejšie prebehnutie s lepšou zložitosťou. Následne máme jeden for cyklus, ktorý bude riadiť aby sme prešli cez všetky joby. Zoberieme si index prvého jobu (s najvyšším profitom). Nasleduje vnútorný for cyklus, kde nájdeme najvyšší možný voľný slot pre tento job. Ten zistíme ako minimum z dealine-1 aktuálneho jobu a n-1 (min(thisJob.deadline - 1, n - 1)). Následne ideme od zadu (to znamená že sa snažíme použiť čo najposlednejší slot pre daný job) a zistíme či je daný slot voľný a ak je priradíme tento job a vyjdeme s vnútorného cyklu. Ak nie, tak sa ho snažíme dať skorej. Ak sa nezmestí alebo sa nedá dať skorej tak sa vnútorný cyklus skončí. Časová zložitosť dvoch vnorených cyklov je Big O(n^2). Tieto dva cykli nám zabezpečia že budeme mať najvyšší možný profit, ktorý sa dal dosiahnuť. Následne funkcia vráti pomocou referencie optimálny rozpis (optimal schedule) pre dané joby a vypíše na obrazovku. Ako testovanie som použil tabuľku zo zadania. Výsledná zložitosť celého programu je teda max(O(n*log(n), O(n^2))) čo je Big O(n^2).

Poznámka: Implementácia Quick Sortu je zo stránky https://www.geeksforgeeks.org/quick-sort/ a bola upravená podľa potrieb.

Úloha 2

Algoritmus na plánovanie s termínmi (scheduling with deadlines algorithm) pri použití dátovej štruktúry Disjoint Set funguje nasledovne:

(Rovnaká časť ako v úlohe 1)
Máme vstupný neusporiadaný vektor prác (jobov), kde každá má pridelené svoje ID číslo, deadline a jej profit. Následne zavoláme funkciu na plánovanie. Funkcia najprv tento neusporiadaný vektor

usporiada podľa profitov zostupne (to znamená že prvý job bude s najvyšším profitom, druhý s nižším atď.). Na usporiadanie som použil Quick Sort algoritmus. Quick sort funguje na princípe divide and conquer (rozdeľuj a panuj). Zvolíme si pivot, ktorý dáme na jeho finálne miesto kde sa bude nachádzať vo výslednom poli a následne rekurzívne voláme quick sort, kde pošleme ľavú časť poľa od pivotu a pravú časť poľa od pivotu. Jeho najhoršia časová zložitosť (Worst case time complexity) je síce Big O(n^2) avšak pre malé datasety, ktoré sú vo forme poľa má priemernú (aj najlepšiu) časovú zložitosť Big O(n * log(n)). Worst case môže nastať pri zlom zvolení pivotu. Ak však pivot zvolíme tak že je približne v polovici výsledného poľa, tak nám to skoro zaručuje jeho rýchlejšie prebehnutie s lepšou zložitosťou.

(rozdielna časť)
Zavoláme funkciu na zistenie najväčšej deadline. Keďže tam je iba jeden cyklus tak zložitosť funkcie je iba Big O(n). Následne si vytvoríme DisjointSet štruktúru. Tá sa inicializuje že sa vytvorený n+1 množín, s tým že n množín je pokrytých n prvkami a jedna množina je naviac. Funkcia find vie nájsť či sa nachádza daný prvok v množine. Funkcia merge vykoná operáciu union nad množinami tz., spojí dve podmnožiny do jednej množiny tak že jedna bude „parentom“ druhej. Po vytvorení štruktúry si inicializujeme vektor kde si uložíme naše práce ako by malo ísť postupne. Potom prejdeme ku jadru programu. Máme tam jeden cyklus, ktorá sa vykoná n krát, pričom n je veľkosť strany matice. Vo vnútri sa nájde najväčší voľný slot pre daný job. Ak nájde voľný slot, tak spojí aktuálny slot s predchádzajúcim. Je to spôsobené tým, že keď je úloha naplánovaná v určitom slote, tento slot a všetky sloty po ňom už nie sú k dispozícii. Následne vypíše joby, ktoré vybralo a sú zoradené podľa profitu. Nakoniec priradí dané joby do vektora a vypíše ich ako majú ísť postupne podľa deadlinov.

Keďže dátová štruktúra je vo forme stromov, tak to znamená že tam platia rovnaké zložitosti ako nad operáciami so stromami. V našom prípade keď vyhľadávame v strome tak zložitosť je Big O(log(n)) a keďže nám toto vyhľadávanie riadi cyklus so zložitosťou Big O(n) ale táto operácia je vnorená tak musíme zložitosti vynásobiť a vyjde nám že celý algoritmus má zložitosť Big O(n*log(n)). Ak berieme do úvahy že priemerná zložitosť Quick Sortu je tiež Big O(n*log(n)), ktorá je dosiahnuteľná v mojom programe tak maximum z týchto dvoch zložitostí nám dá výslednú zložitosť celého programu čo je Big O(n*log(n)).

Poznámka: Implementácia DisjointSet štruktúry a jadra algoritmu (hlavný for cyklus) je zo stránky https://www.geeksforgeeks.org/job-sequencing-problem-using-disjoint-set/ a bola upravená podľa potrieb.

Úloha 3a

Na algorithmus, ktorý rieši problém priradenia prác k pracovníkom (Assignment problem) som použil Greedy (pažravú) metódu. Funguje nasledovne:

Máme maticu n*n, kde sú zapísané jednotlivé ceny práce (jobov) pre daných pracovníkov (workerov). Následne zavoláme funkciu na optimálne priradenie. Inicializujeme vektor, kde bude uložené naše výsledné priradenie. Máme dva for cykly. Jeden vonkajší a jeden vnútorný. Vonkajší zabezpečuje aby sme prešli cez všetkých pracovníkov. Vnútorný cyklus zabezpečuje aby sme prešli cez všetky ceny práce daného pracovníka. Pažravo vyberieme tú najnižšiu cenu a následne si zapíšeme pre ktorú prácu táto cena bola. Takto to ide dokola s tým, že vo vnútri cyklu sa overuje či už nebola daná práca priradená. Takto pažravo nám funkcia vráti sub-optimálne riešenie. Následne si priradenie vypíšeme.

Keďže ide o maticu n*n a aby sme prešli takúto maticu a pozreli sa na všetky možnosti potrebujeme dva cykli s tým, že jeden je vnorený v druhom. Toto nám dáva časovú zložitosť Big O(n^2)

Úloha 3b

Maďarskú metódu som nestihol implementovať keďže som sa zasekol na pokrytí núl s najmenším počtom vertikálnych alebo horizontálnych čiar. Potom mi už neostal čas z dôvodu iných zadaní na iné predmety.

Porovnanie 3a a 3b

Aj keď som neimplementoval maďarskú metódu, tak som zistil pomocou internetu, že jeho časová zložitosť je Big O(n^3). Jeho zložitosť je teda v porovnaní s greedy metódou vyššia avšak vždy nám vráti optimálne riešenie, pričom greedy metóda nám vráti iba sub-optimálne alebo lokálne optimálne riešenie. Inou možnosťou je použiť tzv. brute force metódu (metóda hrubou silou) avšak tá má zložitosť až Big O(n!). Je to z dôvodu že prechádza úplne celú maticu o n krát pri prvej iterácii a nájde najmenšiu cenu a následne sa matica zmenší o jedna a pokračuje v hľadaní v matici o veľkosti n-1 a tak ďalej. Z toho nám vyplýva náš faktoriál. V porovnaní s brute force metódou však obe metódy, ktoré som spomenul majú oveľa nižšiu zložitosť takže by sme mali používať jednu z nich.