Fakulta informatiky a informačných technológií STU, Bratislava Ilkovičova 2, 842 16 Bratislava 4

Praktické zadanie

Analýza a zložitosť algoritmov

Frederik Duvač
3. semester
Date: 5.12. 2023

Obsah

1. Technologické pozadie

1.1. Hardware

Zariadenie, na ktorom boli skompilované zdrojové kódy a testované programy, je notebook Lenovo Legion S7, 12. generácia Intel® Core™ i5-12500H, 16 GB RAM.

1.2. Programovací jazyk

Všetky programy boli implementované v programovacom jazyku C++ vo verzii 17. Použité vývojové prostredie bolo CLion od spoločnosti JetBrains.

2. Implementácie úloh

2.1. Scheduling with deadlines

2.1.1. Implementácia algoritmu

Štruktúra Job predstavuje prácu s tromi atribútmi: id (identifikátor práce), deadline (termín úlohy) a profit (zisk z úlohy). Funkcia `compareJobs()` je pomocná porovnávacia funkcia pre `sort()` funkciu z knižnice `algorithm`, ktorá slúži na zoradenie úloh zostupne podľa zisku.

Funkcia `scheduleJobs()` prijíma počet úloh n a vektor úloh ako vstup. Úlohy sú zoradené zostupne podľa zisku pomocou funkcie `sort()` a vlastnej porovnávacej funkcie `compareJobs()`. Inicializujú sa dva vektory `result` na uchovanie optimálnej sekvencie úloh a `scheduleJob` na sledovanie, či je úloha naplánovaná. Následne sa prechádza úlohami a pre každú úlohu sa iteruje cez možné časové úseky (v opačnom poradí), aby sa našiel dostupný slot pred jej termínom. Ak je slot dostupný, označí úlohu ako naplánovanú, aktualizuje výsledok a preruší cyklus. Nakoniec zobrazí optimálnu sekvenciu úloh a celkový dosiahnutý zisk. V hlavnej funkcii main je inicializovaný vektor s detailmi úloh.

Job	Deadline	Profit
1	2	40
2	4	15
3	3	60
4	2	20
5	3	10
6	1	45
7	1	55

Tabuľka 1. Zoznam úloh, deadlinov a príslušných profitov

2.1.2. Výstup programu

Výstup programu obsahuje optimálnu sekvenciu úloh s maximálnym ziskom a celkový zisk dosiahnutý.

```
Optimal sequence of jobs with maximum profit is: 7 1 3 2
Total profit from jobs is: 170$
```

Obr. 1. Výstup algoritmu Scheduling with deadlines

2.2. Scheduling with Deadlines using Disjoint Set Data Structure III

2.2.1. Use of Disjoint Set Data Structure III

Disjunktný Systém Množín (DSM) je kľúčovou súčasťou algoritmu, ktorý umožňuje efektívne sledovať a aktualizovať plán úloh vzhľadom na ich termíny. Funkcia `findSet()` nájde zastúpenie množiny, ku ktorej patrí prvok, pričom využíva techniku kompresie cesty pre rýchlejšie vyhľadávanie. Funkcia `unionSets()` vykonáva operáciu zjednocovania množín podľa hodnosti, pričom zabezpečuje, aby výsledná množina obsahovala najmenší prvok.

2.2.2. Implementácia modifikácie

Modifikácia spočíva v inicializácii d+1 disjunktných množín, kde d predstavuje maximálny termín úloh. Polia `parent`, `depth` a `smallest` sú následne vytvorené na správne sledovanie a aktualizáciu DSM počas procesu plánovania. Pole `smallest` uchováva informácie o najmenšom prvku v každej množine, čo je dôležité pre optimálne plánovanie úloh tak, aby boli dokončené čo najneskôr, ale stále do ich termínu. Modifikovaný algoritmus následne aktualizuje množiny v priebehu procesu plánovania úloh, spájajúcich tam, kde je to potrebné.

2.2.3. Analýza Modifikovaného Algoritmu

Funkcia `sort()` zabezpečuje triedenie úloh v zostupnom poradí podľa zisku. Časová zložitosť triedenia je O(n log n), kde n je počet úloh.

Inicializácia polí parent, depth, a smallest má časovú zložitosť O(d), kde d je maximálny termín úlohy.

Iterácia cez úlohy (jobs) zabezpečuje plánovanie úloh do optimálnych termínov s využitím DSM. Vnútorný cyklus, ktorý hľadá najnovší voľný časový slot pre úlohu, môže mať najviac O(d) iterácií. Zložitosť tejto časti je O(n * d), kde n je počet úloh.

Celková časová zložitosť je najviac O(n * log n) v dôsledku sortovania a iterácie cez úlohy.

2.2.4. Výstup programu

Algoritmus je navrhnutý na maximalizáciu celkového zisku vzhľadom na termíny úloh. Zobrazuje optimálny postup plánovania úloh s maximálnym ziskom. Vstupné práce, termíny a zisky sú odvodené z Tabuľky 1., aby demonštrovali správne fungovanie algoritmu.

```
Optimal sequence of jobs with maximum profit is: 7 1 3 2
Total profit from jobs is: 170$
```

Obr. 2. Výstup algoritmu Scheduling with deadlines

2.3. Job assignments

2.3.1. Greedy approach

Štruktúra `Assignment` reprezentuje pridelenie úlohy s informáciami o osobe, úlohe a nákladoch.

Funkcia `wasJobAssigned()` kontroluje, či už bola úloha pridelená, prehľadávaním existujúcich pridelení. Funkcia `greedyAssignment()` prechádza každou osobou a priraďuje im úlohu s najnižšími nákladmi (greedy prístup). Nepriradzuje úlohu, ktorá už bola pridelená inému členovi.

Hlavná funkcia `main()` definuje maticu nákladov reprezentujúcu náklady na pridelenie každej osobe k úlohe. Volá funkciu `greedyAssignment()` na nájdenie optimálneho pridelenia. Zobrazuje pridelenia úloh a celkové minimálne náklady. Vypíše optimálne pridelenie úloh pre každú osobu spolu s minimálnymi celkovými nákladmi.

	J1	J2	J3
P1	10	5	5
P2	2	4	10
P3	5	1	7

Tabuľka 2. Príklad cien vykonávania prác jednotlivých osôb

2.3.2. Výstup greedy approach

```
Job assignment:
Person P1 -> Job J2 with cost 5 time units
Person P2 -> Job J1 with cost 2 time units
Person P3 -> Job J3 with cost 7 time units

Minimal assignment job cost: 14 time units
```

Obr. 3. Výstup algoritmu prideľovania úloh použitím greedy approach

2.3.3. Dynamic programming approach

Na úvod sa vytvorí matica `dp` na reprezentáciu všetkých možných priradení prác. Každý prvok `dp[i][j]` je štruktúra `Assignment` s osobou `i`, prácou `j` a príslušnou cenou práce. Matica `dp` sa naplní štruktúrami `Assignment` pre každú kombináciu osoby a práce.

Pre každú osobu sa zoradia práce podľa ich ceny vzostupne pomocou funkcie `sort()` a `compareJobs()`.

Následne sa zoradí samotná maticu dp podľa ceny najefektívnejšej práce pre každú osobu pomocou funkcie `compareAssignmentsCost()`.

Inicializuje sa prázdny vektor assignments na uloženie konečných priradení. Iteruje sa triedenou maticou dp, aby sa našli správne priradenia. Pre každú osobu sa vyberie práca s najnižšou cenou, ktorá ešte nebola priradená. Ak bola práca už priradená vyberá sa nasledujúca najefektívnejšia práca pre danú osobu.

Pre krajší výpis sa zoradí výsledný vektor assignments podľa osôb pomocou funkcie `comparePersons()`.

Vypíšu sa priradené práce pre každú osobu, vrátane príslušnej ceny práce a vypíše sa celková minimálna cena na priradenie prác.

Program v podstate využíva kombináciu triedenia a backtrackingu na nájdenie optimálneho priradenia, zabezpečujúc, aby každá osoba bola priradená k práci s minimálnym nákladom a aby žiadna práca nebola priradená viac ako jednej osobe.

2.3.4. Výstup dynamického programovania

```
Job assignment:
Person P1 -> Job J3 with cost 5 time units
Person P2 -> Job J1 with cost 2 time units
Person P3 -> Job J2 with cost 1 time units

Minimal assignment job cost: 8 time units
```

Obr. 4. Výstup algoritmu prideľovania úloh použitím greedy approach

2.3.5. Porovnanie zložitostí oboch prístupov

2.3.5.1. Greedy approach

Časová zložitosť tohto programu závisí predovšetkým od veľkosti matice nákladov, ktorá predstavuje počet osôb a úloh. Označme n ako veľkosť matice (rozmery n x n).

Funkcia `wasJobAssigned()`, má časovú zložitosť O(n), pretože prechádzame existujúce pridelenia.

Funkcia `greedyAssignment()`, kde vnútorný cyklus pre každú osobu má časovú zložitosť O(n), pretože prechádzame všetky úlohy. Celková časová zložitosť tejto funkcie je O(n^2).

Celková časová zložitosť programu je teda dominovaná časovou zložitosťou funkcie `greedyAssignment()`, čo je O(n^2).

2.3.5.2. Dynamic programing approach

Odhad časovej zložitosti krokov programu:
Inicializácia matice dp: O(n^2), kde n je počet osôb (riadkov) v matici.
Naplnenie matice dp: O(n^2), pretože prechádzame všetky prvky v matici nákladov.
Zoradenie prác pre každú osobu: O(n * m * log(m)), kde n je počet osôb a m je počet prác (stĺpcov) v matici. Sortovanie sa vykonáva pre každú osobu.
Zoradenie osôb podľa najefektívnejšieho nákladu na prácu: O(n * log(n)), kde n je počet osôb. Sortovanie je vykonávané pre všetky osoby.
Nájdenie správnych priradení: O(n * m), pretože pre každú osobu sa prechádza cez možné práce.
Zoradenie priradení podľa osôb: O(n * log(n)), kde n je počet osôb. Sortovanie je vykonávané na konci.

Celková odhadovaná časová zložitosť by mohla byť teda približne O(n^2 + n * m * log(m) + n * log(n)), kde n je počet osôb a m je počet prác.

2.3.6. Zhodnotenie

Greedy prístup je relatívne rýchly a jednoduchý pretože vyberá lokálne optimálne riešenia v každom kroku, ale nie vždy poskytuje globálne optimálne riešenie. Naopak, dynamické programovanie zaručuje optimálny výsledok, ale môže byť výpočtovo náročnejšie ale pre väčšie datasety dokáže nájsť optimálnejšie riešenia. Výber medzi nimi závisí od konkrétnej povahy problému a požiadaviek na výkon.

3. Zdroje

1. C++ sort() function - https://cplusplus.com/reference/algorithm/sort/
2. Neapolitan, Richard. Foundations Of Algorithms. Sudbury : Jones and Bartlett Publishers, Inc., 2014. Vol. 5th.