Fakulta informatiky a informačných technológií

Slovenská technická univerzite v Bratislave

Practical implementation and analysis of an algorithm

Analýza a zložitosť algoritmov

Tomáš Brček

Cvičenie: Utorok 10:00

2023/2024

Tomáš Brček
ID: 120761

Obsah

Tomáš Brček
ID: 120761

Úloha 1 – analýza

void ordering(int n, int deadline[], int profits[], int jobs[])

Časová zložitosť tejto funkcie je *$O(n^2)$,* kde n je počet prvkov v poliach *deadline*, *profits* a *jobs*. Táto funkcia implementuje insertion sort na usporiadanie prvkov v poliach podľa hodnôt v poli *profits* od najväčšieho po najmenší.

Vnútorný while cyklus, ktorý presúva prvky na správne pozície, môže mať v najhoršom prípade lineárnu zložitosť *O(n).* Tento while cyklus sa vykonáva pre každý prvok v poli, ktorých je n, takže celková zložitosť je ***$O(n^2)$***.

void orderByDeadlines(int n, int* K, const int deadline[])

Časová zložitosť tejto funkcie je *$O(n^2)$*, kde n je počet prvkov v poliach *K* a *deadline*. Podobne ako v predchádzajúcej funkcií, implementuje insertion sort na usporiadanie prvkov v poliach *ds* a *K* na základe hodnôt v poli *ds*.

Táto časová zložitosť vyplýva z dvoch vnorených cyklov. Prvý cyklus, ktorý napĺňa pole *ds*, má časovú zložitosť *O(n)*. Druhý cyklus, ktorý implementuje triedenie vkladaním, má časovú zložitosť *$O(n^2)$*, pretože pre každý prvok v poli prechádzame cez všetky predchádzajúce prvky.

Celková zložitosť je teda $O(n) + O(n^2) = \boldsymbol{O(n^2)}$

bool isFeasible(int n, int *K, const int deadline[])

Časová zložitosť tejto funkcie je *O(n)*, kde n je počet prvkov v poliach *K* a *deadline*. Táto funkcia obsahuje jediný for cyklus, ktorý prechádza cez všetky prvky poľa K.

Pretože for cyklus prechádza cez každý prvok práve raz, celková časová zložitosť je lineárna, teda ***O(n)***.

void schedule(int n, const int deadline[], int* J[])

V tejto funkcií sú volané viaceré ďalšie funkcie, ktorých časovú zložitosť je potrebné poznať.

- orderByDeadlines: *$O(n^2)$*
- isFeasible: *O(n)*
- copy: *O(n)*

V každej iterácii cyklu sa volá funkcia *copy*, ktorá má časovú zložitosť *O(n)*. Okrem toho, v každej iterácii sa volá funkcia *orderByDeadlines*, ktorá má časovú zložitosť *$O(n^2)$*, a funkcia *isFeasible*, ktorá má časovú zložitosť *O(n)*.

Celkový počet iterácií cyklu je najviac n.

Pre každú iteráciu cyklu sa vykonávajú operácie s lineárnou časovou zložitosťou a operácie s kvadratickou zložitosťou. Preto celková časová zložitosť funkcie *schedule* bude

$$O(n) \times \left(O(n) + O(n^2) + O(n^2)\right) = \boldsymbol{O(n^3)}$$

int main()

Vo funkcií *main* sú volané funkcie *ordering* a *schedule*. Na základe týchto funkcií vieme určiť aj celkovú časovú zložitosť funkcie *main*, ako aj celého programu.

- Funkcia *ordering* používa algoritmus triedenia vkladaním, ktorý má časovú zložitosť *$O(n^2)$*, kde n je počet prvkov vstupného poľa.
- Funkcia *schedule* volá funkcie *copy*, *orderByDeadlines*, a *isFeasible* v cykle, ktorý má maximálne n iterácií. Pre každú iteráciu sa vykonávajú operácie s lineárnou a kvadratickou zložitosťou. Celková časová zložitosť funkcie *schedule* je *$O(n^3)$* vzhľadom na zložitosť funkcií, ktoré volá.

Celkovo je teda časová zložitosť daná najväčšou časovou zložitosťou z funkcií *ordering* a *schedule*. Preto môžeme povedať, že časová zložitosť funkcie *main* je ***$O(n^3)$*** v najhoršom prípade, kde n je počet jobov.

Celková časová zložitosť programu bude teda ***$O(n^3)$***, keďže časová zložitosť funkcie *schedule* je najvyššia.

Tomáš Brček
ID: 120761

Úloha 2 – analýza

Opis kódu

Opis kódu je poskytnutý v kóde v podobe komentárov.

void makeset(index i)

Operácie vo vnútri funkcie majú konštantný čas, a preto je časová zložitosť tejto funkcie ***O(1)***.

void add_job(index i, int job)

Operácie vo vnútri funkcie majú konštantný čas, a preto je časová zložitosť tejto funkcie ***O(1)***.

void merge(set_pointer p, set_pointer q)

Funkcia obsahuje if-else blok, pričom každý blok obsahuje while cyklus. While cyklus iteruje, kým *U[p].parent* nie je rovnaký ako p. V rámci while cyklu sú konštantné operácie: priradenia a porovnania. Funkcia obsahuje tiež konštantné operácie mimo while cyklu.

V najhoršom prípade while cyklus prechádza predkov danej množiny, kým nedosiahne koreň. Hĺbka takéhoto stromu môže byť najviac *O(log m)*, kde m je maximum z poľa *deadline*. Preto celková časová zložitosť funkcie je v najhoršom prípade ***O(log m)***.

int small(set_pointer p)

Operácie vo vnútri funkcie majú konštantný čas, a preto je časová zložitosť tejto funkcie ***O(1)***.

int partition(int deadline[], int profits[], int jobs[], int low, int high)

Časová zložitosť tejto funkcie závisí od počtu operácií vykonaných v cykle, ktorý prechádza cez prvky a porovnáva ich s pivotom. Keďže každý prvok je porovnávaný najviac raz a presunutý najviac raz, časová zložitosť tejto časti je **O(n)**, kde n je počet prvkov v poli.

void quickSort(int deadline[], int profits[], int jobs[], int low, int high)

QuickSort algoritmus rozdeľuje pole okolo pivota a potom rekurzívne triedi obidve polovice. Pre každé rekurzívne volanie sa pole delí na polovicu, čo dáva logaritmickú zložitosť. Vzhľadom na to, že v každom deliacom kroku je vykonaná operácia *partition()* s časovou zložitosťou *O(n)*, celková časová zložitosť *quickSort()* je **O(n log n)**, kde n je počet prvkov v poli.

void ordering(int n, int deadline[], int profits[], int jobs[])

Táto funkcia jednoducho volá quickSort() na zadané polia. Preto je časová zložitosť tejto funkcie tiež **O(n log n)**, kde n je počet prvkov v poli.

void schedule(int n, int *total_profit, int deadline[], int profit[])

Časová zložitosť tejto funkcie je *O(n log m)*, kde n je počet prvkov v poli a m predstavuje maximum z deadlinov.

Cyklus for prechádza cez všetky prvky poľa, kde n je počet prvkov. Obsahuje operácie s konštantným časom (porovnania, priradenia, volanie funkcie *small*, podmienené vetvy).

While cyklus sa opakuje najviac n-krát (v najhoršom prípade). Obsahuje operácie s konštantným časom (porovnania, priradenia, podmienené vetvy).

Vo funkcii sa nachádza aj volanie funkcie *merge*, ktorej časová zložitosť je *O(log m)*, ako je opísané vyššie.

Celková časová zložitosť je daná počtom iterácií cyklu for, čo je n, násobené časovou zložitosťou funkcie merge. Celková časová zložitosť je ***O(n log m)***.

int main()

Časová zložitosť funkcie main závisí od časovej zložitosti operácií, ktoré sa v nej vykonávajú. Funkcia *main* obsahuje niekoľko volaní iných funkcií (*makeset*, *ordering*, *schedule*).

Volanie makeset: *O(1)*, čo je konštantná operácia.

Volanie ordering: *O(n log n)*, kde n je počet prvkov v poli.

Volanie schedule: *O(n log d)*, kde n je počet prvkov v poli a d je maximum z poľa deadline

$$O = \max\{O(1), O(n\ log\ n), O(n \log m)\}$$

Celková časová zložitosť funkcie main je potom obmedzená najvyšším časom trvania z týchto volaní, teda ***O(n log m)***, kde n je počet prvkov v poli a m je maximum z d.

Celková časová zložitosť programu bude teda ***O(n²)***, keďže časová zložitosť funkcie *ordering* je najvyššia.

Tomáš Brček
ID: 120761

Úloha 3a – analýza

int findMax(int table[][N])

Časová zložitosť tejto funkcie je ***O(n^2)***, kde n je veľkosť matice. Funkcia obsahuje dva vnorené cykly, ktoré prechádzajú všetky prvky matice s rozmerom "n x n". Pre každý prvok matice vykonáva porovnanie s aktuálnym maximom, takže celkový počet porovnaní má kvadratickú zložitosť.

void findMinimum(int table[][N], int used[][N], int* x, int* y, int previous, int max)

Časová zložitosť tejto funkcie je ***O(n^2)***, kde "n" je veľkosť matice. Funkcia opäť obsahuje dva vnorené cykly, ktoré prechádzajú všetky prvky matice s rozmerom "n x n". V každom kroku porovnáva hodnoty v matici a aktualizuje minimálnu hodnotu a príslušné súradnice, ak sú splnené určité podmienky. Celkový počet operácií v tomto prípade je tiež kvadratický vzhľadom na veľkosť matice.

bool isFeasible(int n, int *K)

Časová zložitosť tejto funkcie je *O(n^2)*, kde n je veľkosť poľa K. Funkcia obsahuje dva vnorené cykly, pričom vnútorný cyklus začína na indexe i+2 a postupuje o 2 kroky. To znamená, že vnútorný cyklus prechádza iba každý druhý prvok poľa, pričom pre každý prvok vykonáva porovnanie s prvkom na inom mieste v poli.

Celkový počet iterácií vonkajšieho cyklu je n. Celkový počet iterácií vnútorného cyklu je n/4, keďže začína na indexe i+2 a postupuje o 2 kroky až po n. Keďže konštantné členy môžeme vynechať, tak časová zložitosť tejto funkcie je ***O(n^2)***.

void assign(int table[][N], int* J[], int max)

Keďže v algoritme sú použité funkcie *findMinimum*, *copy*, a *isFeasible*, je potrebné poznať zložitosti týchto funkcií:

- findMinimum: *O(n^2)*.
- copy: ide o obyčajné kopírovanie prvkov poľa, jej časová zložitosť je *O(n).*
- isFeasible: *O(n^2).*

Vieme, že zložitosť týchto funkcií bude rozhodujúca pre celkovú časovú zložitosť algoritmu. Samotný algoritmus obsahuje cyklus, ktorý prebehne v najhoršom prípade až n^2-krát. V tomto cykle sú volané vyššie spomenuté funkcie. Pre zložitosť tejto funkcie je určujúca konštrukcia if, kde sa volá funkcia *isFeasible()* s časovou zložitosťou *O(n^2)*. V prípade splnenia podmienky sa vykonáva telo tejto podmienky, kde je volaná funkcia *copy* s časovou zložitosťou *O(n)*. Na základe toho môžeme povedať, že časová zložitosť algoritmu bude ***O(n^4)***.

int main()

Časová zložitosť funkcie main závisí od časovej zložitosti funkcií, ktoré sa v nej vykonávajú. Funkcia *main* obsahuje niekoľko volania iných funkcií (*findMax, assign*).

Volanie findMax: *O(n^2)*, kde n je veľkosť matice.

Volanie assign: *O(n^4)*, kde n je veľkosť matice.

$$O = \max\{O(n^2), O(n^4)\}$$

Celková časová zložitosť funkcie *main* je potom obmedzená najvyšším časom trvania z týchto volaní, teda $\boldsymbol{O(n^4)}$.

Celková časová zložitosť programu bude teda $\boldsymbol{O(n^4)}$, keďže časová zložitosť funkcie *assign* je najvyššia.

Tomáš Brček
ID: 120761

Úloha 3b – analýza

int findMinInRow(int matrix[][N], int i)

Zložitosť tejto funkcie je ***O(n)***, kde n je počet stĺpcov v matici. For-cyklus prechádza cez všetky prvky v zadanom riadku a porovnáva ich s aktuálnym minimom. Vzhľadom na to, že každý prvok v riadku porovnáva práve raz, celkový počet operácií v tejto funkcii rastie lineárne.

int findMinInCol(int matrix[][N], int j)

Zložitosť tejto funkcie je ***O(n)***, kde n je počet riadkov v matici. For-cyklus prechádza cez všetky prvky v zadanom stĺpci a porovnáva ich s aktuálnym minimom. Vzhľadom na to, že každý prvok v stĺpci porovnáva práve raz, celkový počet operácií v tejto funkcii rastie lineárne.

void subtractRow(int matrix[][N], int min, int i, int rowsZeros[])

Časová zložitosť tejto funkcie je ***O(n)***, kde n predstavuje počet stĺpcov v matici. For-cyklus prechádza cez všetky prvky v zadanom riadku matice a vykonáva konštantný počet operácií pre každý prvok. Časová zložitosť lineárne závisí od počtu stĺpcov.

void subtractCol(int matrix[][N], int min, int j, int rowsZeros[])

Časová zložitosť tejto funkcie je ***O(n)***, kde n predstavuje počet riadkov v matici. For-cyklus prechádza cez všetky prvky v zadanom stĺpci matice a vykonáva konštantný počet operácií pre každý prvok. Časová zložitosť lineárne závisí od počtu riadkov.

bool hasZero(int mask[][N], int j)

Časová zložitosť tejto funkcie je ***O(n)***, kde n predstavuje počet riadkov v matici. For-cyklus prechádza cez všetky prvky v zadanom stĺpci matice a vykonáva konštantný počet operácií pre každý prvok. Zložitosť je lineárna.

bool mask_zeros(int rowsZeros[], int matrix[][N], int mask[][N])

V tejto funkcií sa nachádza viacero vnorených cyklov.

Prvý cyklus prechádza všetky riadky a v každom riadku prechádza všetky stĺpce. Pre každý riadok a stĺpec sa vykoná konštantný počet operácií, a to až do počtu stĺpcov n. Celkový počet operácií v tomto cykle je teda $O(n^2)$.

Druhý cyklus prechádza všetky stĺpce a v každom stĺpci prechádza všetky riadky. Rovnakým spôsobom, pre každý stĺpec a riadok sa vykoná konštantný počet operácií, a to až do počtu riadkov n. Celkový počet operácií v tomto cykle je tiež $O(n^2)$.

Celková časová zložitosť tejto funkcie je teda $\max\{O(n^2), O(n^2)\}$ = $\boldsymbol{O(n^2)}$

void assignJobs(int assignment[], int matrix[][N], int rowsZeros[])

Časová zložitosť tejto funkcie je $O(n^2)$, kde n je rozmer matice. Prechádzame všetky prvky matice vo vnorenom cykle. Pre každý prvok vykonáva konštantný počet operácií.

Celkový počet operácií je teda $\boldsymbol{O(n^2)}$.

int main()

Prvý for-cyklus: V tomto cykle sa od každého prvku v každom riadku odpočíta minimum z daného riadka. V tomto cykle sú volané funkcie *findMinInRow* a *subtractRow* s časovou zložitosťou *O(n)*. Keďže sú volané vo vnútri cyklu, tak jeho časová zložitosť je *O(n^2)*.

Druhý for-cyklus: : V tomto cykle sa od každého prvku v každom stĺpci odpočíta minimum z daného stĺpca. V tomto cykle sú volané funkcie *findMinInCol* a *subtractCol* s časovou zložitosťou O(n). Keďže sú volané vo vnútri cyklu, tak jeho časová zložitosť je *O(n^2)*.

Cyklus while: Tento cyklus beží, kým na pokrytie všetkých núl nie je potrebných n čiar. V každom kroku tohto cyklu sa vyhľadá minimum nepokrytých prvkov, a potom sa toto minimum odčíta od všetkých nepokrytých prvkov a pripočíta k prvkom, ktoré sú pokryté dvakrát. V najhoršom prípade by sa tento proces mohol opakovať *O(n^2)*-krát, čo dáva celkovú časovú zložitosť *O(n^3)*.

assignJobs: *O(n^2)*

Celková časová zložitosť funkcie main:

$$\max\{O(n^2), O(n^2), O(n^3), O(n^2)\} = \boldsymbol{O(n^3)}$$

Celková časová zložitosť programu je potom ***O(n^3)***.

Porovnanie 3a a 3b

Časová zložitosť:

- Greedy approach: ***O(n^4)***
- Dynamic programming: ***O(n^3)***

Greedy prístup a dynamické programovanie predstavujú dva odlišné prístupy k riešeniu problému priradenia osôb k úlohám, pričom každý má svoje výhody a nevýhody.

Greedy prístup, implementovaný v programe 3a, sa vyznačuje jednoduchosťou s časovou zložitosťou **O(n^4)**. Tento prístup však nemusí vždy dosahovať globálne minimum a môže byť citlivý na začiatočné podmienky.

Naopak, dynamické programovanie v programe 3b sa snaží nájsť optimálne riešenie prostredníctvom systematického prehľadávania rôznych kombinácií. Jeho časová zložitosť je **O(n^3)**. Avšak, dynamické programovanie môže byť náročnejšie na pamäť, a teda môže byť menej efektívne pre veľmi veľké vstupy.

Analýza ukázala, že program 3a využíva hrubý prístup s kvadratickou zložitosťou vo funkcii *assign*, zatiaľ čo program 3b využíva Maďarský algoritmus na dosiahnutie optimálneho riešenia s kubickou zložitosťou.

Tomáš Brček
ID: 120761

<u>Zhrnutie</u>

V prvej úlohe praktického zadania išlo o implementáciu algoritmu na riešenie problému *„Scheduling with deadlines“*. V mojej implementácií tohto algoritmu sa mi podarilo dosiahnuť zložitosť **O(n^3)**.

Druhá úloha bola modifikácia prvej úlohy, kde sme takisto mali vyriešiť problém *„Scheduling with deadlines“*. V tomto riešení sme však museli použiť dátovú štruktúru *Disjoint set*, vďaka ktorej sme pridávali úlohy, čo najneskôr to bolo možné. Na základe analýzy tohto algoritmu, ktorá bola poskytnutá vyššie, má moja implementácia algoritmu časovú zložitosť **O(n log m)**.

Tretia úloha sa skladá z dvoch častí, v ktorých máme poskytnúť implementáciu riešenia *problému priradenia*. V prvej časti úlohy bolo potrebné implementovať riešenie tohto problému pomocou *Greedy approach*. V tejto implementácií má moje riešenie zložitosť **O(n^4)**. V druhej časti tejto úlohy sme mali problém implementovať pomocou *dynamického programovania*, pričom som využil *Hungarian algorithm*. Táto implementácia dosahuje zložitosť **O(n^3)**.

<table>
<tr><td>Úloha 1</td><td rowspan="2">Scheduling with deadlines</td><td>Greedy approach</td><td>O(n^3)</td></tr>
<tr><td>Úloha 2</td><td>Disjoint set</td><td>O(n log m)</td></tr>
<tr><td rowspan="2">Úloha 3</td><td rowspan="2">Assignment problem</td><td>Greedy approach</td><td>O(n^4)</td></tr>
<tr><td>Dynamic programming</td><td>O(n^3)</td></tr>
</table>