Šeste vaje APS2: dinamično programiranje, 2. del

Naloga 1: barvanje grafov

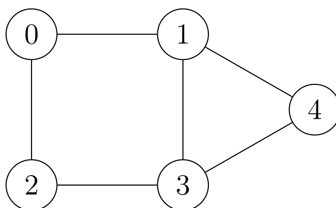
Opis problema

Podan je neusmerjen graf G. Njegova vozlišča želimo pobarvati tako, da sosednji vozlišči ne bosta iste barve. Najmanjšemu številu barv, ki ga potrebujemo za to opravilo, rečemo *kromatsko število* grafa in ga označimo s $\chi(G)$.

Problem barvanja grafa (iskanja $\chi(G)$) je v splošnem NP-težak, zato zanj po vsej verjetnosti ni učinkovite rešitve. Če se ga lotimo naivno, potrebujemo $O(mn^n)$ časa, kjer je n število vozlišč, m pa število povezav. Število možnih barvanj vozlišč s k barvami znaša k^n, za preverjanje veljavnosti barvanja pa moramo preveriti vse povezave. Kromatsko število je v najslabšem primeru enako številu vozlišč.

Ideja

Naj $G[U]$ označuje podgraf grafa G, ki vsebuje vozlišča $U \subseteq V$ in vse povezave $(u, v) \in E$, za katere velja $u \in U$ in $v \in U$. Na primer, za graf na sliki 1 je graf $G[\{1, 2, 4\}]$ sestavljen iz vozlišč 1, 2 in 4 ter povezave (1, 4).



Slika 1: Primer grafa.

Spomnimo se tudi, da je *neodvisna množica* grafa podmnožica množice vozlišč, tako da noben par vozlišč v njej ni med seboj povezan. Na primer, ena od neodvisnih množic grafa na sliki 1 je množica $\{2, 4\}$.

Kromatsko število lahko določimo nekoliko učinkoviteje, če upoštevamo, da lahko vozlišča v neodvisni množici grafa pobarvamo z isto barvo. Če iz grafa G odstranimo neodvisno množico I in ugotovimo, da lahko preostanek grafa ($G[V \backslash I]$) pobarvamo s k barvami, potem vemo, da lahko graf G pobarvamo s $k + 1$ barvami. Seveda moramo izbrati takšno neodvisno množico I, da bo vsota $1 + \chi(G[V \backslash I])$ minimalna:

$$\chi(G) = \begin{cases} 0, & \text{če } G \text{ nima vozlišč;} \\ 1 + \min_{I \in \mathcal{I}(G)} \chi(G[V \backslash I]) & \text{sicer,} \end{cases}$$

pri čemer je $\mathcal{I}(G)$ množica vseh nepraznih neodvisnih množic grafa G.

Na primer, graf na sliki 1 ima devet nepraznih neodvisnih množic: $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{0, 3\}$, $\{0, 4\}$, $\{1, 2\}$, $\{2, 4\}$. Če iz grafa odstranimo množico $\{0\}$, potem lahko preostanek grafa pobarvamo s tremi barvami. Vendar pa to ne pomeni, da je kromatsko število grafa enako $3 + 1 = 4$; če odstranimo, na primer, množico $\{2, 4\}$, lahko preostanek grafa (torej graf z vozlišči 0, 1 in 3 ter povezavama $(0, 1)$ in $(1, 3)$) pobarvamo z dvema barvama. Izkaže se, da je to optimalna izbira. Kromatsko število grafa je torej enako 3.

Algoritem

Najprej se moramo dogovoriti, kako učinkovito predstavljamo podmnožice množice vozlišč (zaradi enostavnosti bomo predpostavili, da so vozlišča grafa označena z 0, 1, ..., $n - 1$).

Saj že vemo: množico $\{a_1, a_2, \ldots, a_k\}$ predstavimo z n-bitnim številom, v katerem so biti na indeksih a_1, a_2, $\ldots$, a_k enaki 1, ostali pa 0. Na primer, če je $n = 5$, bomo množico $\{0, 2, 3\}$ predstavili z dvojiškim številom 01101 (kot ponavadi je bit z indeksom 0 najlažji, torej skrajno desni bit).

Druga lastnost, ki jo opazimo, je tale: pri računanju kromatskega števila nekega grafa bomo uporabljali kromatska števila njegovih podgrafov, pri čemer lahko isti podgraf nastopa kot sestavni del več različnih večjih grafov. To pomeni, da imamo prekrivajoče se podprobleme, zato se problema lotimo z dinamičnim programiranjem. Kot vemo, pri dinamičnem programiranju potrebujemo shrambo že izračunanih rezultatov, ki jo lahko polnimo bodisi »od zgoraj navzdol« (z rekurzijo in memoizacijo) ali »od spodaj navzgor« (iterativno). Omenjena shramba bo v našem primeru kar tabela velikosti 2^n (število podmnožic množice $\{0, \ldots, n-1\}$), v kateri bo element na indeksu i hranil že izračunano kromatsko število grafa, določenega z množico vozlišč, ki ga predstavlja število i. V primeru na sliki 1 bo kromatsko število grafa na množici vozlišč $\{0, 2, 3\}$ shranjeno na indeksu $01101_{(2)} = 13_{(10)}$.

Pri iterativnem polnjenju shrambe že izračunanih rezultatov moramo paziti na pravilen vrstni red računanja: ko računamo kromatsko število grafa, določenega z množico vozlišč $U \subseteq V$, morajo biti kromatska števila za vse njegove podgrafe (torej za vse podmnožice množice U) že izračunana. To lahko dosežemo preprosto tako, da tabelo polnimo po naraščajočih indeksih, saj imajo vse podmnožice množice U manjši indeks kot množica U.

Primer

Slika 2 prikazuje delovanje algoritma za določanje kromatskega števila na grafu s slike 1. Pripravimo si tabelo T z 2^5 elementi in jo polnimo po naraščajočih indeksih; spomnimo se, da v element na indeksu i vpišemo kromatsko število grafa na množici vozlišč, ki jo predstavlja indeks i. Nekaj izračunov si oglejmo podrobneje:

- Element $T[0]$ se nanaša na graf brez vozlišč. Njegovo kromatsko število je enako 0.
- Element $T[1]$ se nanaša na graf na vozlišču $\{0\}$. Ta ima eno samo neprazno neodvisno množico — množico $\{0\}$. Kromatsko število grafa potemtakem izračunamo takole:

$$\begin{aligned} T[1] &= \chi(G[\{0\}]) \\ &= 1 + \min\{\chi(G[\{0\} \backslash \{0\}])\} \\ &= 1 + \min\{\chi(G[\varnothing])\} \\ &= 1 + \min\{T[0]\} \\ &= 1. \end{aligned}$$

- Element $T[3]$ se nanaša na graf na vozliščih $\{0, 1\}$, ki ima dve neprazni neodvisni množici: $\{0\}$ in $\{1\}$.

$$\begin{aligned} T[3] &= \chi(G[\{0,1\}]) \\ &= 1 + \min\{\chi(G[\{0,1\} \backslash \{0\}]),\ \chi(G[\{0,1\} \backslash \{1\}])\} \\ &= 1 + \min\{\chi(G[\{1\}]),\ \chi(G[\{0\}])\} \\ &= 1 + \min\{T[2], T[1]\} \\ &= 2. \end{aligned}$$

- Graf na množici $\{0, 3\}$ ima tri neprazne neodvisne množice: $\{0\}$, $\{3\}$ in $\{0, 3\}$.

$$T[9] = \chi(G[\{0,3\}])$$

$$
\begin{aligned}
&= 1 + \min\{\chi(G[\{0,3\}\backslash\{0\}]),\ \chi(G[\{0,3\}\backslash\{3\}]),\ \chi(G[\{0,3\}\backslash\{0,3\}])\} \\
&= 1 + \min\{\chi(G[\{3\}]),\ \chi(G[\{0\}]),\ \chi(G[\emptyset])\} \\
&= 1 + \min\{T[8], T[1], T[0]\} \\
&= 1.
\end{aligned}
$$

- Celoten graf (tj. graf na množici $\{0,1,2,3,4\}$) ima devet nepraznih neodvisnih množic: $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{0,3\}$, $\{0,4\}$, $\{1,2\}$ in $\{2,4\}$.

$$
\begin{aligned}
T[31] &= \chi(G[\{0,1,2,3,4\}]) \\
&= 1 + \min\{\chi(G[\{0,1,2,3,4\}\backslash\{0\}]),\ \chi(G[\{0,1,2,3,4\}\backslash\{1\}]), \\
&\qquad \chi(G[\{0,1,2,3,4\}\backslash\{2\}]),\ \chi(G[\{0,1,2,3,4\}\backslash\{3\}]), \\
&\qquad \chi(G[\{0,1,2,3,4\}\backslash\{4\}]),\ \chi(G[\{0,1,2,3,4\}\backslash\{0,3\}]), \\
&\qquad \chi(G[\{0,1,2,3,4\}\backslash\{0,4\}]),\ \chi(G[\{0,1,2,3,4\}\backslash\{1,2\}]), \\
&\qquad \chi(G[\{0,1,2,3,4\}\backslash\{2,4\}])\} \\
&= 1 + \min\{\chi(G[\{1,2,3,4\}]),\ \chi(G[\{0,2,3,4\}]),\ \chi(G[\{0,1,3,4\}]), \\
&\qquad \chi(G[\{0,1,2,4\}]),\ \chi(G[\{0,1,2,3\}]),\ \chi(G[\{1,2,4\}]), \\
&\qquad \chi(G[\{1,2,3\}]),\ \chi(G[\{0,3,4\}]),\ \chi(G[\{0,1,3\}])\} \\
&= 1 + \min\{T[30], T[29], T[27], T[23], T[15], T[22], T[14], T[25], T[11]\} \\
&= 3
\end{aligned}
$$

0	1	2	3	4	5	6	7
0	1	1	2	1	2	1	2
$\{\}$	$\{0\}$	$\{1\}$	$\{0,1\}$	$\{2\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$

8	9	10	11	12	13	14	15
1	1	2	2	2	2	2	2
$\{3\}$	$\{0,3\}$	$\{1,3\}$	$\{0,1,3\}$	$\{2,3\}$	$\{0,2,3\}$	$\{1,2,3\}$	$\{0,1,2,3\}$

16	17	18	19	20	21	22	23
1	1	2	2	1	2	2	2
$\{4\}$	$\{0,4\}$	$\{1,4\}$	$\{0,1,4\}$	$\{2,4\}$	$\{0,2,4\}$	$\{1,2,4\}$	$\{0,1,2,4\}$

24	25	26	27	28	29	30	31
2	2	3	3	2	2	3	3
$\{3,4\}$	$\{0,3,4\}$	$\{1,3,4\}$	$\{0,1,3,4\}$	$\{2,3,4\}$	$\{0,2,3,4\}$	$\{1,2,3,4\}$	$\{0,1,2,3,4\}$

Slika 2: Delovanje algoritma za določanje kromatskega števila na grafu s slike 1.

Časovna zahtevnost

Za vsako od 2^n podmnožic U množice V in za vsako neodvisno množico $I \subseteq U$ moramo v izračunati množico $U \backslash I$ in jo uporabiti kot indeks v tabelo. Če neodvisne množice grafa poiščemo vnaprej (to lahko storimo z naivnim postopkom) in če za delo z množicami uporabljamo učinkovite operacije, lahko časovno zahtevnost algoritma ocenimo kot

$O(2^n 2^n) = O(4^n)$. Obstajajo tudi učinkovitejši algoritmi, a zaenkrat vsi tečejo v času $\Omega(2^n)$.

2-obarvljivost grafov

Če nas zanima, ali lahko graf pobarvamo z dvema barvama, lahko dobimo odgovor v času $O(m)$, in sicer s preprostim iskanjem v globino ali širino. Premislite sami!

Naloga 2: maksimalna vsota v matriki

Podana je matrika $n \times n$ z elementi $a_{ij} \in \mathbb{Z}_0^+$ ($i, j \in \{0, \ldots, n-1\}$). Naj bo σ permutacija množice $\{0, \ldots, n-1\}$ (bijektivna preslikava množice same vase). Definirajmo

$$V(\sigma) = \sum_{i=0}^{n-1} a_{i,\sigma(i)}$$

in zasnujmo algoritem, ki poišče

$$V^* = \max_{\sigma \in \mathcal{S}_n} V(\sigma),$$

kjer $\mathcal{S}_n$ označuje množico vseh permutacij množice $\{0, \ldots, n-1\}$. Vrednost V^* je torej največja možna vsota n elementov matrike, pri čemer v vsaki vrstici in vsakem stolpcu izberemo natanko en element.

Na primer, za matriko

$$A = \begin{bmatrix} 13 & 7 & 12 & 6 \\ 9 & 1 & 16 & 14 \\ 15 & 2 & 10 & 3 \\ 5 & 4 & 11 & 8 \end{bmatrix}$$

je $V^* = 7 + 14 + 15 + 11 = 47$, pripadajoča permutacija pa števila 0, 1, 2, 3 preslika v števila 1, 3, 0, 2. □

Naivni pristop (preizkus vseh permutacij) nam dá časovno zahtevnost $O(n!\, n) = O((n+1)!)$, s premislekom, kakršnega smo že vajeni, pa lahko ta rezultat nekoliko izboljšamo. Kako bi izračunali maksimalno vsoto elementov matrike, pri čemer moramo v vsaki vrstici in vsakem stolpcu izbrati natanko en element? Recimo takole:

- V prvi vrstici izberemo nek element, denimo tistega v stolpcu j.
- Nato izračunamo vrednost V^* za podmatriko, ki sega od druge do zadnje vrstice, pri čemer ne smemo izbrati nobenega elementa iz stolpca j.
- Indeks j v prvem koraku izberemo tako, da maksimiziramo vsoto izbranega elementa v prvem koraku in vrednosti V^* za podmatriko v drugem koraku.

Pri rekurzivnem razcepu moramo poleg indeksa prve vrstice podmatrike, ki jo trenutno obravnavamo, vzdrževati še množico stolpcev, iz katerih še lahko črpamo elemente. Naj bo $v(i, S)$ maksimalna vsota podmatrike, ki se prične na indeksu i in pri kateri lahko izbiramo elemente iz stolpcev, določenih z množico indeksov S. To vrednost lahko izračunamo takole:

$$v(i, S) = \begin{cases} 0 & \text{pri } i = n; \\ \max_{j \in S}(a_{ij} + v(i+1, S \backslash \{j\})) & \text{sicer.} \end{cases}$$

Rešitev izhodiščnega problema je $v(0, \{0, 1, \ldots, n-1\})$.

Sedaj že vemo, da lahko rekurenčno enačbo razmeroma enostavno pretvorimo v memoiziran rekurziven ali iterativen algoritem. Pri obeh bi si pripravili tabelo D velikosti $n \times 2^n$.

Celica $D[i, b(S)]$, kjer je $b(S)$ desetiška vrednost karakterističnega vektorja množice S, bi hranila vrednost $v(i, S)$. Tak pristop bi vodil do prostorske zahtevnosti $O(2^n n)$ in časovne zahtevnosti $O(2^n n^2)$, saj bi za vsako celico tabele D potrebovali še sprehod po enicah v karakterističnem vektorju množice S.

No, če se osredotočimo na iterativno rešitev, lahko prihranimo še nekaj malega prostora in časa:

- Prvič, vrednosti $v(i, .)$ so odvisne le od vrednosti $v(i + 1, .)$, zato zadošča, če v pomnilniku hranimo le trenutno in predhodno vrstico matrike D. (Ker pri iterativnem algoritmu matriko D polnimo po *padajočih* indeksih vrstice, ima »trenutna« vrstica indeks i, »predhodna« pa indeks $i + 1$.)
- Drugič, vrednost $v(i, S)$ izračunamo samo za množice S, ki imajo natanko $n - i$ elementov. Podmatrika, ki se prične z vrstico z indeksom i, ima namreč $n - i$ elementov, zato bomo v njej izbrali natanko $n - i$ stolpcev.

Na ta način zmanjšamo prostorsko zahtevnost na $O(2^n)$. Časovna zahtevnost ostaja $O(2^n n^2)$, a sodeč po eksperimentih se vsaj konstantni faktor nekoliko zmanjša.

Pri APS2 nas implementacijske podrobnosti v splošnem ne zanimajo, zato algoritme pišemo v psevdokodi. Tokrat bomo naredili izjemo. Skrivnostna funkcija `__builtin_popcount` je dodatek prevajalnika GCC, ki vrne število enic v bitni predstavitvi podanega števila.

```
int poisci(const vector<vector<int>>& matrika) {
    int n = matrika.size();
    int dvaNaN = 1 << n;

    // podmnozicePoVelikostih[i]: predstavitve podmnožic velikosti i;
    // npr. pri n = 4 imamo podmnozicePoVelikostih[2] = [3, 5, 6, 9, 10, 12]

    vector<vector<int>> podmnozicePoVelikostih(n + 1);
    for (int s = 0; s < dvaNaN; s++) {
        podmnozicePoVelikostih[__builtin_popcount(s)].push_back(s);
    }

    vector<int> prej(dvaNaN);   // D[i + 1, :]

    for (int i = n - 1; i >= 0; i--) {
        vector<int> zdaj(dvaNaN);   // D[i, :]

        for (int s: podmnozicePoVelikostih[n - i]) {
            // sprehodi se po vseh podmnožicah množice S (te so predstavljene z vektorji,
            // ki jih iz vektorja s dobimo tako, da eno enico spremenimo v ničlo)
            int t = s;
            int j = 0;
            while (t > 0) {
                if (t & 1) {    // Je na indeksu j v vektorju s enica?
                    zdaj[s] = max(zdaj[s], matrika[i][j] + prej[s ^ (1 << j)]);
                }
                j++;
                t >>= 1;
            }
        }
        prej = zdaj;
    }
    return prej[dvaNaN - 1];
}
```