Viggo Kann 2022-08-26

DD2350 Algoritmer, datastrukturer och komplexitet

Uppgifter till övning 7

Probabilistiska algoritmer, reduktioner

Skipplistor Sannolikheten p att ett element på nivå i i en skipplista också finns på nivå $i+1$ kan man variera efter behov. På sida 5 i skipplistartikeln analyseras hur sökstigens förväntade längd och det förväntade antalet pekare i ett element beror av p. Där framgår att kortaste sökstigen fås om $p = 1/e \approx 0.3679$. Varför kan det trots detta vara bättre att välja $p = 1/2$ eller $p = 1/4$?

Verifikation av matrismultiplikation Snabbaste kända metoden för multiplikation av två $n \times n$-matriser är $O(n^{2,376})$. Visa att det går att probabilistiskt verifiera en matrismultiplikation i kvadratisk tid. Närmare bestämt, konstruera en Montecarloalgoritm som i tid $O(kn^2)$ verifierar att $AB = C$ och har en felsannolikhet på högst 2^{-k}.

Reduktion som ger negativt resultat Att hitta medianelementet (det $\lceil n/2 \rceil$e minsta elementet) av n positiva heltal kräver minst $2n$ jämförelser i värsta fallet. Utnyttja detta resultat för att bestämma en undre gräns för antalet jämförelser som krävs för att hitta det $\lceil n/2 \rceil + 10$e minsta elementet.

Reduktion som ger positivt resultat Ett ofta användbart sätt att lösa problem på är att hitta en reduktion till ett problem som man redan vet hur man ska lösa. Du ska nu använda denna metod för att lösa kantsammanhängandegradsproblemet som definieras på följande sätt.

> INMATNING: En sammanhängande oriktad graf $G = (V, E)$ och ett positivt heltal K mellan 1 och $|V|$.
>
> PROBLEM: Är det tillräckligt att ta bort K kanter från grafen G för att göra den osammanhängande (dvs uppdelad i flera sammanhängande komponenter)?

Formulera optimeringsproblem som beslutsproblem Jobbmaskningsproblemet definieras som ett optimeringsproblem på följande sätt:

Indata: Arbetsdagens längd T och n arbetsuppgifter. Uppgift i tar tiden t_i och kräver arbetsinsatsen w_i. Alla tal är positiva heltal.

Lösning: Ett urval av arbetsuppgifterna $D \subseteq [1..n]$ som tillsammans fyller ut en arbetsdag, det vill säga $\sum_{i \in D} t_i \geq T$.

Målfunktion: Den arbetsinsats som krävs för att utföra dagsverket D, alltså $\sum_{i \in D} w_i$.

Mål: Minimera målfunktionen, alltså hitta det dagsverke som kräver minst arbetsinsats.

a) Formulera optimeringsproblemet som ett beslutsproblem. Utvidga indata med ett målvärde I för arbetsinsatsen och formulera en fråga som har svaret ja eller nej.

b) Turingreducera optimeringsproblemet till beslutsproblemet, det vill säga visa hur man kan lösa optimeringsproblemet med hjälp av en algoritm som löser beslutsproblemet.

Reduktioner mellan besluts-, optimerings- och konstruktionsproblem Anta att algoritmen GraphColouring(G,k) på tid $T(n)$ (där n är antalet hörn i G) svarar 1 omm hörnen i G kan färgas med k färger utan att någon kant har likfärgade ändpunkter.

a) Konstruera en algoritm som givet en graf G med n hörn bestämmer det minimala antalet färger som behövs för att färga G. Tiden ska vara $O(\log n \cdot T(n))$.

b) Konstruera en algoritm som givet en graf G med n hörn färgar hörnen med minimalt antal färger i tid $O(P(n)T(n))$ där $P(n)$ är ett polynom.

Lösningar

Lösning till Skipplistor
Det beror på att man vill snabbt kunna beräkna RandomLevel, alltså på vilken nivå ett nytt element ska in. Om $p = 2^{-k}$ så kan detta göras effektivt. Man läser helt enkelt k bitar av en slumpsträng (en ström av bitar) och kollar till exempel om alla k bitarna är 1. □

Lösning till Verifikation av matrismultiplikation
Idé: slumpa fram en n-vektor $\mathbf{r} \in \{0,1\}^n$, beräkna $\mathbf{a} = C \circ \mathbf{r}$ och $\mathbf{b} = A \circ (B \circ \mathbf{r})$ och verifiera att $\mathbf{a} = \mathbf{b}$.

Om $AB = C$ så blir alltid $\mathbf{a} = \mathbf{b}$. Om $AB \neq C$ så är $D = AB - C \neq 0$. Då måste det finnas något index i i $\mathbf{r}$ som påverkar värdet av $D\mathbf{r}$ om man byter $\mathbf{r}[i]$ så att det blir 1 om det var 0 och 0 om det var 1. Alltså, om algoritmen slumpar fram elementen i vektorn $\mathbf{r}$ med sannolikhet $1/2$ för 0 och $1/2$ för 1 så kommer $AB\mathbf{r} \neq C\mathbf{r}$ med sannolikhet minst $1/2$. Om vi upprepar detta k gånger kommer sannolikheten för att algoritmen blir lurad alla k gångerna vara mindre än $(1/2)^k$.

Algoritmen blir alltså:

```
VerifyMult(n, A, B, C, k) =
        for i ← 1  to k do
            r ← slumpvektor∈ {0,1}^n
            a ← C ∘ r
            b ← A ∘ (B ∘ r)
            if a ≠ b then return false
        return true
```

Tidskomplexiteten blir $O(kn^2)$ eftersom algoritmen gör k varv och i varje varv görs tre matris-vektormultiplikationer (i själv verket är det bara additioner eftersom vektorn är binär) som tar $O(n^2)$. □

Lösning till Reduktion som ger negativt resultat
Reducera medianproblemet till det nya problemet genom att lägga till så många ettor till indata så att det tidigare medianelementet nu är det $\lceil m/2 \rceil + 10$e minsta elementet, där m är antalet element efter det att ettorna lagts till. Det betyder att vi måste lägga till 20 ettor och $m = n + 20$. Den undre gränsen $2n$ blir därmed uttryckt i m: $2n = 2m - 40$. □

Lösning till Reduktion som ger positivt resultat
Först bör vi försöka förstå problemet. Anta att X är en minimal kantmängd vars borttagande gör G osammanhängande. (Detta ska tolkas som att ingen strikt delmängd av X uppfyller detsamma.) Då gäller att G uppdelas i två komponenter, och alla kanterna i X går mellan dessa två. (Annars skulle ju X inte vara minimalt.) X svarar alltså mot ett snitt i grafen (en uppdelning av hörnmängden i två delar). Antalet kanter i X kan sägas vara snittets storlek. Detta innebär att det minsta antalet kanter som vi måste ta bort för att G ska bli osammanhängande — kalla detta $\lambda(G)$ — är lika med storleken på ett minsta snitt (V_1, V_2) i G.

Minimalt snitt är ju samma som maximalt flöde. Vi ska därför försöka reducera kantsammanhängandegradsproblemet till maximalt flöde mellan två hörn s och t i en graf. Om vi utökar G till en flödesgraf G' genom att ge varje kant dubbelriktad kapacitet 1, så kan vi alltså finna $\lambda(G)$ genom att beräkna maxflödet från s till t för olika s och t. Vi behöver dock inte variera båda dessa. Om vi väljer s godtyckligt så måste det höra till någon av mängderna V_1 och V_2 ovan. Genom att variera t över alla hörn utöver s kommer så vi garanterat att träffa något hörn i den andra mängden. Slutsatsen blir alltså att för ett godtyckligt hörn s gäller

$$\lambda(G) = \min_{t \in V - \{s\}} \{\text{MaxFlow}(G', s, t)\}.$$

Om vi ska vara noggranna så var nu uppgiften att avgöra om $K \geq \lambda(G)$. Vi kan alltså svara NEJ om $K < \text{MaxFlow}(G', s, t)$ för alla $t \in V - \{s\}$ och JA annars.

Flödesalgoritmen anropas $|V| - 1$ gånger. Varje flödesberäkning kan göras i tid $O(|V|^3)$ (vilket man inte behöver kunna utantill). Alltså blir komplexiteten för vår algoritm $O(|V|^4)$. □

Lösning till Formulera optimeringsproblem som beslutsproblem

a) **Indata:** Arbetsdagens längd T, målet I och n arbetsuppgifter. Uppgift i tar tiden t_i och kräver arbetsinsatsen w_i.

Fråga: Finns det en delmängd $D \subseteq [1..n]$ så att $\sum_{i \in D} t_i \geq T$ och $\sum_{i \in D} w_i \leq I$?

b) Anta att Jobbmaskning$(T, I, n, \{t_i\}, \{w_i\})$ är en algoritm som löser beslutsproblemet. Optimala totala arbetsinsatsen måste vara ett heltal mellan 0 och $\sum_{1 \leq i \leq n} w_i$. Gör en binärsökning mellan dessa extremvärden efter det minsta värde I som Jobbmaskning$(T, I, n, \{t_i\}, \{w_i\})$ svarar ja för.

□

Lösning till Reduktioner mellan besluts-, optimerings- och konstruktionsproblem

a) Vi vet att antalet färger är mellan 1 och n. Använd binärsökning i detta intervall för att med hjälp av algoritmen GraphColouring hitta ett k så att GraphColouring$(G, k) = 1$ och GraphColouring$(G, k-1) = 0$. Detta innebär nämligen att minimala färgningen har k färger. Det behövs $\log n$ halveringar av n för att komma ner till 1. Tidskomplexiteten blir därför $O(\log n \cdot T(n))$.

b) Hitta först det minimala antalet färger k med metoden ovan. Vi vill färga hörnen i G med färger med nummer 1 till k. Följande algoritm gör det:

CreateColouring$(G = (V, E),\ k)$=
$u \leftarrow$ första hörnet i V
$C \leftarrow \{u\}; u.colour \leftarrow k$
foreach $v \in V - \{u\}$ **do**
  if $(u, v) \notin E$ **then**
    if GraphColouring$((V, E \cup \{(u, v)\}), k) = 1$ **then** $E \leftarrow E \cup \{(u, v)\}$
    else $C \leftarrow C \cup \{v\}; v.colour \leftarrow k$
if $k > 1$ **then** *CreateColouring*$((V - C, E), k - 1)$

GraphColouring anropas högst en gång för varje par av hörn i grafen. Tidskomplexiteten för hela algoritmen blir därför $O(\log n \cdot T(n) + n^2 \cdot T(n)) = O(n^2 \cdot T(n))$. □