Övningstillfälle 1: Programmeringsövningar · 1MA020

Vilhelm Agdur[1]

[1] vilhelm.agdur@math.uu.se

22 februari 2023

Detta dokument innehåller en samling övningar i kombinatorik som kräver programmering: Vi utforskar alltså några olika saker med hjälp av stora mängder beräkningar.

Träd och Prüferkoder

Övning 1. Skriv en funktion som, givet ett träd,[2] räknar ut dess Prüferkod.

[2] Använd förslagsvis ett färdigt paket för grafer för att representera ditt träd, så att du får funktioner för att plotta din graf gratis.

Övning 2. Skriv en funktion som, givet en Prüferkod, genererar dess korresponderande träd.

Med dessa funktioner skrivna har vi nu ett naturligt sätt att generera ett slumpmässigt träd på n noder, genom att generera en slumpmässig Prüferkod.

Övning 3. Skriv en funktion som genererar ett slumpmässigt träd på n noder med denna metod, och bevisa att denna kommer generera varje träd med samma sannolikhet.[3]

[3] Här har vi alltså lite bevis i vår programmering – vilket väl är naturligt, om vi vill vara säkra på att vår kod fungerar.

Det här är inte den enda naturliga metoden för att generera ett slumpmässigt träd. Låt oss också studera en annan modell för ett slumpmässigt träd.

Övning 4. Antag att X är en slumpvariabel som tar värdet i med sannolikhet p_i, för varje $i \in \mathbb{Z}^{\geq 0}$.

Vi skapar oss ett slumpmässigt träd genom att tänka oss att vi startar med en individ, som sedan får ett antal barn som dras från fördelningen till X. Sedan får varje av dessa barn också ett antal barn givet av denna fördelning, oberoende av varandra och sina förfäder.

Skriv en funktion som, givet en annan funktion som genererar ett slumptal,[4] genererar ett slumpmässigt träd på detta vis. Ett träd av detta slag kallas för ett *Galton-Watson-träd*.

[4] Om ni skriver i ett språk som varken har funktioner som första klassens objekt eller pekare till funktioner får ni nog ta vektorn $(p_0, p_1, \ldots)$ som argument i stället.

Övning 5. Givet ett rotat träd T kan vi kan definiera en slumpvariabel X genom att slumpmässigt välja en nod i trädet, och låta X vara antalet barn till denna nod.

Generera ett par olika slumpmässiga träd likformigt, med metoden med Prüferkoder, och generera sedan ett Galton-Watson-träd med hjälp av det X som motsvarar dessa träd. Rita upp träden sida vid sida. Ser de ut som ungefär samma träd eller är de radikalt annorlunda?

Övning 6. Låt oss definiera ett par olika parametrar för ett träd:

1. Dess *gradföljd* $(d_1, d_2, \ldots, d_n)$ ges av att d_i är antalet grannar till nod i.
2. Dess *genomsnittliga grad* $\bar{d}$ är genomsnittet av antalet grannar hos en nod i trädet, det vill säga $\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$.
3. Dess *maximala* och *minimala* grad är maximum respektive minimum av d_i.
4. Dess *diameter* är det största avståndet mellan två noder i trädet, där vi med avståndet mellan u och v menar att om vi har en följd av distinkta noder $u = v_0\, v_1\, v_2 \ldots v_{k-1}\, v_k = v$ sådan att v_{i-1} och v_i är grannar för varje i så är avståndet mellan dem k.[5]

Låt nu X vara en Poissonfördelad slumpvariabel med parameter $\frac{3}{4}$,[6] och låt T_{GW} vara ett Galton-Watson-träd med X som fördelning för antalet barn. Låt T_{Unif} vara ett likformigt slumpmässigt valt träd på lika många noder som T_{GW}.

Simulera dessa par av träd ett stort antal gånger, och för varje av de parametrar vi definierade ovan, rita upp ett histogram över dess värde för de två olika träden. Inkludera medelvärdet för parametern för de två olika träden i rubriken till histogrammet.

[5] I ett träd finns det ju alltid exakt en sådan följd, så att denna definition är otvetydig. I en allmän graf menar vi längden på den *kortaste* sådana följden.

[6] Det vill säga,

$$\mathbb{P}(X = k) = \frac{(3/4)^k e^{-3/4}}{k!}.$$

Oberoende mängder i grafer och den giriga algoritmen

I vårt bevis av Caro-Weis sats om oberoende mängder i grafer finns det implicit en algoritm för att hitta en sådan: Välj en slumpmässig etikettering av grafen, och låt sedan vår oberoende mängd vara mängden av noder som har en lägre etikett än alla sina grannar.

Det finns också en relaterad, ännu enklare algoritm, den så kallade giriga algoritmen: Givet en etikettering av din graf, gå igenom dess noder i ordning från minst till störst. Varje gång du ser en nod som inte har några grannar redan i din oberoende mängd, lägg till noden du nu är på till din oberoende mängd.

Övning 7. Bevisa att du, ifall du tar din slumpmässiga ordning från Caro-Wei-algoritmen och i stället kör den giriga algoritmen, alltid kommer att få en oberoende mängd som innehåller den som Caro-Wei gav dig, men som kan vara större.

Övning 8. Implementera dessa bägge algoritmer.

Övning 9. Vi vet från föreläsningarna att en Erdős-Rényi-graf $G(n, p)$ har n noder, och varje möjlig kant är med i grafen oberoende med sannolikhet p. Skriv en funktion som givet n och p genererar en sådan slumpmässig graf.

Övning 10. Generera ett stort antal $G\left(10000, \frac{15}{10000}\right)$-grafer. För varje av dem, kör Caro-Wei-algoritmen, den giriga algoritmen med den slumpmässiga ordning ni använde för Caro-Wei, och den giriga algoritmen i ordning från lägst grad till högst. Kalla storlekarna på de oberoende mängderna dessa hittar för $\alpha_{CW}(G)$, $\alpha_{GU}(G)$, och $\alpha_{GO}(G)$.

Rita upp ett histogram över varje av dessa tre tal och räkna ut deras medelvärden. Räkna också ut medelvärdet av $\frac{\alpha_{GU}(G)}{\alpha_{CW}(G)}$ – notera att detta *inte* är samma sak som att räkna ut medelvärdena av de två talen och sedan genomföra divisionen.

Bonus: Inga av dessa algoritmer är i närheten av optimala. Kan ni komma på någon algoritm som presterar bättre? Implementera den och se i praktiken hur mycket större mängderna den finner blir än vad den giriga algoritmen finner.