EINFÜHRUNG

Beispiele für randomisierte Algorithmen

1.) der randomisierte QUICKSORT

Eingabe: die Zahlen $S = \{x_1, x_2, x_3 \dots x_n\}$ (verschiedene Zahlen, obdA.)

Aufgabe: sortiere diese Zahlen

QUICKSORT Schema:

→ falls $|S| \leq 1$ gib S aus (schon sortiert)

→ sonst:

- wähle ein Pivot-Element $x \in S$
- vergleiche jedes andere Element mit x und sei S_1 die Teilmenge aller Elemente kleiner als x
  S_2 — " — größer als x
- sortiere S_1 und S_2 mit QUICKSORT, seien $\bar{S}_1$ und $\bar{S}_2$ die sortierten Folgen
- gib $\bar{S}_1, x, \bar{S}_2$ aus

An welcher Stelle benutzt man Randomisierung?

→ das Pivot-Element kann man deterministisch (z.B. das linkeste Element), oder randomisiert/zufällig auswählen (in diesem Fall gemäß Gleichverteilung)

E2.

Wann ist ein Pivot Element schlecht bzw. gut für die Laufzeit?

Nehmen wir an z.B., dass der Alg. deterministisch das linkeste Element als Pivot wählt, und die Eingabe ist $n, n-1, n-2, \ldots, 3, 2, 1$.

Dann gibt es in QUICKSORT $(n-1)+(n-2)+(n-3)+\ldots+2+1 =$

$= \frac{n(n-1)}{2} = \Theta(n^2)$ Vergleiche $\Rightarrow \Theta(n^2)$ ist die Worst-Case Laufzeit

(für den deterministischen Alg. würden wir Worst-Case Eingabe sagen; für den randomisierten Alg. immer zufällig das größte Element zu wählen entspricht einer Worst-Case Berechnung)

Im Idealfall wäre das Pivot-Element so gewählt, dass es die Menge S in zwei gleichgroße Mengen teilt, also x wäre der Median. Wäre dies bei jedem rekursiven Aufruf der Fall, dann hätte jedes Element $y \in S$ maximal $\log_2 n$ Vergleiche bis (es selbst als Pivot gewählt wird, oder) die ihn enthaltende Teilmenge nur noch ein Element enthält.

$\leq n \cdot \log_2 n$

$|S| \; |S_1| \; |S_{11}|$

$(n, \frac{n}{2}, \frac{n}{4}, \ldots, 2, 1)$

$\sim \log_2 n$ mal kann S halbiert werden

in diesem Fall haben wir Laufzeit $O(n \cdot \log n)$

→ Ein Pivot-Element ist also „gut" falls es die Menge S in etwa gleichgroße Teile aufteilt, und ist „schlecht" wenn es unter den kleinsten/größten Elementen von S ist und deshalb nicht gut aufteilt.

Was ist der Vorteil einer zufälligen Pivot-Wahl?

→ Bei der randomisierten Pivot-Wahl ist es unwahrscheinlich, dass „zu oft" „zu schlechte" Pivot-Elemente gewählt werden: man kann zeigen, dass die erwartete Laufzeit (also nicht nur der Idealfall!) $O(n \cdot \log n)$ ist.

Was bedeutet hier erwartete Laufzeit?

die erwartete Laufzeit ist ähnlich zu einer durchschnittlichen Laufzeit, wobei der Durchschnitt über viele unabhängige Abläufe vom randomisierten QUICKSORT für dieselbe Eingabe gedacht ist. Dies entspricht der durchschnittlichen / erwarteten Laufzeit wenn wir „den Durchschnitt" (die Erwartung) über alle möglichen Folgen von gewählten Pivot-Elementen nehmen. Dies weiter entspricht dem Erwartungswert (der Laufzeit) über allen möglichen Folgen der Zufallsbits die der Algorithmus während seiner Berechnung erhält.

Beachte, dass die Erwartung NICHT über die verschiedenen (ohne Permutieren!) Reihenfolgen der Eingabezahlen berechnet wird. Wir machen keine Annahmen über (die Verteilung) dieser) Reihenfolge!

Theorem: Wenn das Pivot-Element jedes mal zufällig gemäß Gleichverteilung gewählt wird, dann ist die erwartete Laufzeit von QUICKSORT für jede fixierte Eingabe S

$2n \ln n + O(n)$. (Beweis im Algorithmentheorie-Skript)

Die Erwartung wird über die Zufallsentscheidungen des Algorithmus genommen.

E 4.

Bemerkung: Eine mathematisch äquivalente Variante des Alg. könnte das Pivot-Element zwar deterministisch (linkeste) wählen, dafür aber die Eingabe $x_1 x_2 \ldots x_n$ am Anfang zufällig permutieren (so dass jede Reihenfolge gleichwahrscheinlich ist). Diese hätte dieselbe erwartete Laufzeit $O(n \cdot \log n)$ mit derselben Analyse, (abgesehen von der Laufzeit der zufälligen Permutation am Anfang). Aus dieser Variante des Algorithmus (die rein theoretisch ist) wird es offensichtlich: Der randomisierte QUICKSORT verwendet Randomisierung um worst-case Eingaben (Folgen wie $n, n-1, \ldots 2, 1$ für die linke Pivot-Wahl) zu vermeiden.

Wie trifft ein Algorithmus zufällige Entscheidungen? Z.B. wie genau wird ein zufälliges Pivot-Element aus n Elementen ausgewählt?

→ $\lceil \log_2 n \rceil$ Zufallsbits ergeben einen Index/eine Position i in $\{1, 2, \ldots, n\}$; falls $n = 2^{\log_2 n} < i \leq 2^{\lceil \log_2 n \rceil}$ kann ein neues i gezogen werden, oder jedes i normiert mit $\frac{n}{2^{\lceil \log n \rceil}} \cdot i$ werden

Was ist ein randomisierter Algorithmus?

Ein randomisierter Algorithmus verfügt über alle Fähigkeiten eines deterministischen Algorithmus, und kann zusätzlich auch Zufallsbits anfordern; er erhält jedes mal das Bit 0 (bzw. 1) mit Wahrscheinlichkeit $\frac{1}{2}$. (Der Alg. wird verschiedenste Berechnungen B, in Abhängigkeit von den erhaltenen Zufallsbits ausführen.)

Wie wird die erwartete Laufzeit definiert?

0 1 1 0 0 0 1 0 0 1 1 0 0 0 0

Jede konkrete Bitfolge der Länge k hat Wahrscheinlichkeit $\left(\frac{1}{2}\right)^k = \frac{1}{2^k} = 2^{-k}$

Definition: Sei A ein randomisierter Algorithmus und w eine Eingabe für A fixiert.

- die Wahrscheinlichkeit einer Berechnung B von A ist $p_B = \frac{1}{2^k}$, falls die Berechnung B k Zufallsbits anfordert.
- sei $Zeit_B$ die Anzahl der Schritte einer Berechnung B. Die erwartete Laufzeit von A auf Eingabe w ist

$$\sum_{B \text{ Berechnung}} p_B \cdot Zeit_B.$$

Beispiel: das Programm hält, sobald das Zufallsbit = 1 (solange werden Bits angefordert) Berechne die erwartete Laufzeit!

Bemerkung: Die Anzahl k der angeforderten Zufallsbits hängt natürlich von der Berechnung selbst (von w und den bisherigen Zufallsbits) ab. In QUICKSORT z.B. hängt die Anzahl der benötigten Pivot-Elemente von den vorherigen Pivot-Elementen ab.
Die Erwartung könnte man auch über gleichlange (hinreichend lange) Bitfolgen berechnen, die alle gleichwahrscheinlich sind. Das Ergebnis für den Erwartungswert wäre dasselbe!

2.) Bestimmung des durchschnittlichen Gehalts, ohne das eigene Gehalt preiszugeben

Gegeben sind

n Personen mit Gehalt G_i für Person i

(G_i ist nur für i bekannt)

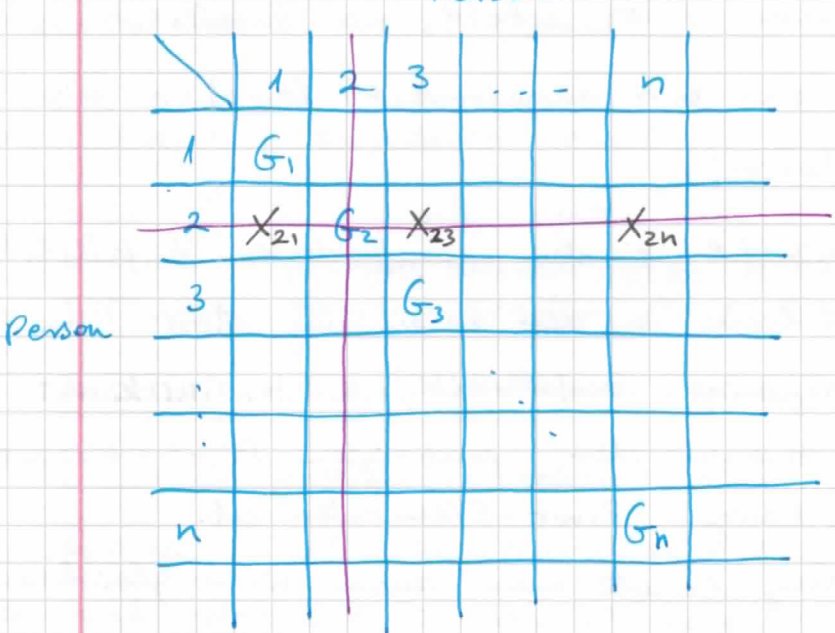
Problem:

Die Personen wollen ihr durchschnittliches Gehalt berechnen, ohne das eigene Gehalt zu verraten.

$$\frac{\sum_{i=1}^{n} G_i}{n}$$ also praktisch $\sum_i G_i$ berechnen

Die Idee eines randomisierten Protokolls

eine imaginäre Tabelle wird „aufgefüllt"



- Wir füllen die Diagonale mit den Werten $G_1\ G_2 - - G_n$

- die restlichen Kästchen werden mit zufällig gewählten Werten x_{ij} ausgefüllt, so dass jede Person ausschließlich ihre eigene Zeile und ihre eigene Spalte kennt.

Genauer:

→ Person i wählt zufällig die Zahlen in ihrer Zeile:

$$x_{i1}\ x_{i2}\ \dots\ x_{i,i-1}\ x_{i,i+1}\ \dots\ x_{in}$$

und teilt jede x_{ij} der Person j mit

→ Jetzt wird die Summe $\sum_{i=1}^{n} G_i$ so berechnet, dass jeder i addiert zu G_i noch alle x_{ij} in ihrer Zeile, und subtrahiert alle x_{ki} in ihrer Spalte. Somit wird jeder Eintrag x_{ij} $i \neq j$ genau einmal zur Summe addiert (von der Person i) und genau einmal subtrahiert (von der Person j)

(Somit würde Person i den Anderen den folgenden Wert $\tilde{S}_i$ mitteilen:

$$\tilde{S}_i = G_i + \sum_{j:\, j\neq i} x_{ij} - \sum_{k:\, i\neq k} x_{ki}$$

z.B.: alle Summanden in $\tilde{S}_2$

$$-x_{12} \quad x_{21} + G_2 + x_{23} + x_{24} \quad -x_{32} \quad -x_{42}$$

und nach Summieren der $\tilde{S}_i$ Werte für alle n Personen

$$\sum_{i=1}^{n} \tilde{S}_i = \sum_{i=1}^{n} G_i + \sum_{i,j:\, i\neq j} x_{ij} - \sum_{i,j:\, i\neq j} x_{ij} = \sum_{i=1}^{n} G_i \quad)$$

E8. Noch genauer:

Das tatsächliche Protokoll:

- Die Werte X_{ij} werden aus einem Intervall $\{0, 1, \dots, M-1\}$ zufällig gleichverteilt gewählt, wobei M groß genug ist so dass $\sum_{i=1}^{n} G_i < M$ sicherlich gilt.

- Von jeder Person i wird der Wert

$$S_i = \left(\left(G_i + \sum_{j:\, j \neq i} X_{ij} - \sum_{k:\, k \neq i} X_{ki} \right) \bmod M \right)$$

bekannt gegeben.

- So gilt $\sum_i S_i \equiv \sum G_i \pmod M$

und wegen $\sum G_i < M$, kann nur

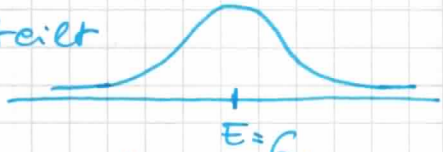
$$\sum S_i \pmod M = \sum G_i$$ gelten.

[Wozu war es gut, dass die S_i modulo M berechnet werden?

So wird erreicht, dass die S_i im Wesentlichen (fast) gleichverteilte Werte (gleichverteilte Zufallsvariablen) über

$\{0, 1, \dots, M-1\}$ sind.

(Ansonsten wäre $\tilde{S}_i$ etwa normalverteilt mit Erwartungswert



$$E = G_i + \frac{(n-1)(M-1)}{2} - \frac{(n-1)(M-1)}{2} = G_i$$)]

Bemerkung: Aus den veröffentlichten Werten $S_2\ S_3 \ldots S_n$ kann Person 1 nichts mehr als die Summe $G_2 + G_3 + \ldots + G_n$ herausfinden.
Ebenso: eine Koalition von k Personen kann höchstens die Summe der Gehälter der restlichen $n-k$ Personen erfahren.

Die Verwendung von Randomisierung in diesem Beispiel entspricht der Entwurfsmethode „Randomisierung in Konfliktsituationen" (gegen Gegner). Die Zufallszahlen haben hier die privaten Werte G_i gedeckt.

Beispiele für Online-Algorithmen

Ein Online Algorithmus erhält eine Folge von Eingaben, und berechnet die i-te Ausgabe (d.h. trifft eine Entscheidung über die i-ten Eingabe) direkt nachdem die i-te Eingabe erhalten wurde, ohne die Eingaben $i+1, i+2, i+3, \ldots$ usw. gesehen zu haben.

1.) Online BIN-PACKING

Objekte mit Gewichten $g_1\ g_2\ g_3 \ldots g_n \ldots$ $(0 \le g_n \le 1)$ kommen online. Wir müssen die Objekte sofort Behältern mit Kapazität 1 zuweisen, so dass möglichst wenige Behälter verwendet werden.
Wir haben also keine Möglichkeit z.B. die Gewichte zu sortieren.
Bekannte online Algorithmen sind z.B. Next Fit, First Fit, oder Best Fit

E 10.

2.) Das Ski-Problem

Wir fahren in den Ski-Urlaub für n Tage.
Wir haben die Möglichkeit, für 1 € / Tag Skier zu leihen, oder für K € Skier zu kaufen ($K < n$).
Wir wissen nicht wie lange es Schnee gibt / wann wir aufgeben (müssen) → d.h. wir kennen unseren letzten Tag nicht. Wann sollen wir Skier kaufen (wenn überhaupt)? Was ist eine gute Strategie um verhältnismäßig wenig Verlust zu machen?

3.) Das Sekretär-Problem

Wir möchten eine Stelle mit einem kompetenten Bewerber besetzen. Wir haben n Bewerber für die Stelle, müssen aber jedem Bewerber direkt nach dem Bewerbungsgespräch zu- oder absagen.

Was ist eine gute Strategie, den besten Bewerber mit nicht-zu-kleiner Wahrscheinlichkeit auszuwählen?

Beachte: der Erfolg ist nicht garantiert: wir werden z.B. wahrscheinlich den besten verpassen, falls er sich als erster vorstellt

Eine „ganz gute" Strategie für das Sekretär-Problem

(online + randomisiert)

Unsere Annahmen:

- die Zahl n der Bewerber ist von vornherein bekannt
- die Bewerber erscheinen in einer zufälligen Reihenfolge zum Bewerbungsgespräch. (wir mischen sie so dass jede Permutation gleichwahrscheinlich ist)
- nach jedem Bewerbungsgespräch können wir dem Bewerber i eine Note N_i geben, die Noten sind alle unterschiedlich (dies vereinfacht die Analyse)

(Nach dem ersten Gespräch kennen wir nur die Note N_1 und haben kein Vergleich mit den Anderen (wir haben keine a-priori Information über die (Verteilung der) Noten!)

Mit der Strategie „nimm den ersten Bewerber" hat man die Wahrscheinlichkeit $\frac{1}{n}$, den Besten zu treffen. Ähnlich schlecht wäre eine Strategie die eine Entscheidung auf die ersten c Bewerbern basiert, für eine Konstante c unabhängig von n. Eine Strategie, die bis zu den letzten paar Bewerbern wartet auf Gewissheit, wäre auch blöd.)

Idee: Man wartet bis man einen bestimmten Bruchteil der Bewerber gesehen hat, ohne zuzusagen.
Danach nimmt man den Ersten, der besser ist als alle in dieser Teilmenge.

E12.

Der Algorithmus (für gegebenen Parameter r oder genauer für $0 < \frac{r}{n} < 1$)

- Wir würfeln die Reihenfolge der Bewerber zufällig (gleichverteilt) aus;
- Wir sagen den ersten r Bewerbern ab, aber merken uns die Höchstnote N unter den ersten r Bewerbern;
- Wir sagen dem ersten verbleibenden Bewerber zu, der eine bessere Note als N hat;
- Falls es keinen solchen gibt, sagen wir dem letzten zu

Wir müssen noch den bestmöglichen Parameter $\frac{r}{n}$ bestimmen!

Warum ist $\frac{r}{n}$ = „sehr klein" schlecht?

→ die untere Schranke N könnte zu klein werden, und wir akzeptieren einen Bewerber mit niedriger Note.

Warum ist $\frac{r}{n}$ = „zu groß" schlecht?

→ wahrscheinlich fällt dann der beste Bewerber unter den ersten r, und wir verpassen ihn.

r $N_1 N_2 \dots N_r$ ist wie eine Stichprobe

Wir berechnen die beste Wahl für $\frac{r}{n}$

Unser Ziel ist es, die Erfolgswahrscheinlichkeit (d.h. dass der Beste gewählt wird) zu maximieren.

- bezeichne b die Position des besten Bewerbers, also N_b sei maximal unter allen Noten
- es gilt $\text{Prob}(b=i) = \frac{1}{n}$ für jede $1 \le i \le n$ weil der Beste mit Wahrscheinlichkeit $\frac{1}{n}$ auf Position i ist in der Reihenfolge der Gespräche
- falls $b \le r$, werden wir keinen Erfolg haben
- falls $b > r$ was ist die Erfolgswahrscheinlichkeit?

Beobachtung: Für jede konkrete $b > r$, wir haben genau dann Erfolg, wenn der-beste-Bewerber-unter-$(N_1, N_2 \dots N_{b-1})$ unter den ersten r Bewerbern ist.

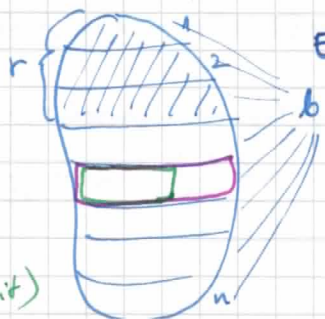
Für jede $r+1 \le i \le n$, die Wahrscheinlichkeit, dass

der Beste an Position i erscheint $(b=i)$, und

der Beste-in-$(N_1, N_2 \dots N_{b-1})$ zu den ersten r gehört, ist

$$\frac{1}{n} \cdot \frac{r}{i-1}$$

(Erfolgswahrscheinlichkeit falls $b=i$ ((bedingte Wahrscheinlichkeit)



Ereignisse = alle Permutationen

E 14.

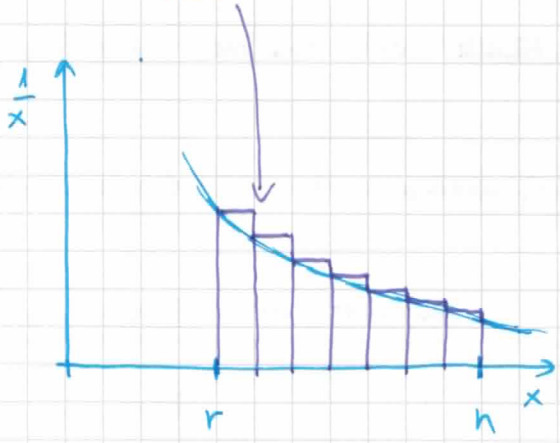
Die Erfolgswahrscheinlichkeit ist

$$p = \sum_{i=r+1}^{n} \frac{1}{n} \cdot \frac{r}{i-1} = \frac{r}{n} \cdot \sum_{i=r+1}^{n} \frac{1}{i-1} = \frac{r}{n} \cdot \sum_{i=r}^{n-1} \frac{1}{i} \approx$$

$$\approx -\frac{r}{n} \cdot \ln \frac{r}{n} = p\left(\frac{r}{n}\right)$$

(✿ wir approximieren $\sum_{i=r}^{n-1} \frac{1}{i}$ durch das Integral $\int_r^n \frac{1}{x}\,dx$

$$\int_r^n \frac{1}{x}\,dx = \ln n - \ln r = -(\ln r - \ln n) = -\ln \frac{r}{n}$$



)

Wir sind am Wert des Bruchs $\frac{r}{n}$ interessiert, der $p\left(\frac{r}{n}\right)$ maximiert. $\quad p\left(\frac{r}{n}\right) = -\frac{r}{n} \cdot \ln \frac{r}{n}$

D. h. maximiere $g(y) = -y \cdot \ln y$ über $y \in [0,1]$

(wobei $y = \frac{r}{n}$)

$$g'(y) = -\ln y - 1$$

$$g'(y) = 0 \iff \ln y = -1$$

$$\Updownarrow$$

$$y = \frac{1}{e}$$

$\Rightarrow$ das optimale Verhältnis ist $\frac{r}{n} = \frac{1}{e}$

die Erfolgswahrscheinlichkeit ist dann

$$p\left(\frac{1}{e}\right) = -\frac{1}{e} \ln \frac{1}{e} = -\frac{1}{e} \cdot (-1) = \frac{1}{e}$$

Wir sollen also die ersten $\frac{n}{e}$ Bewerber ablehnen, und dann den ersten nehmen der besser ist als all diese. Die Erfolgswahrscheinlichkeit (dass der Beste gewählt wird) ist dann $p \approx \frac{1}{e}$

(zumindest ist diese eine konstante Wahrscheinlichkeit -- besser als $\frac{1}{n}$)

Wie beim randomisierten QUICKSORT, bezieht sich auch hier die Wahrscheinlichkeit auf die Gleichverteilung der Reihenfolge der Bewerber (bei QUICKSORT, Zahlen) nach Permutationen. Diese wird durch die Zufallsentscheidungen (Bits) des Algorithmus erzeugt. Also, Erwartungswert Wahrscheinlichkeit, usw. werden bezüglich der Zufallsschritte des Algorithmus definiert, und gelten für jede Eingabe.

[Die Analysen wären analog im Fall eines deterministischen Algorithmus, wenn man annehmen würde, dass die Reihenfolge der Zahlen / Bewerber zufällig gleichverteilt ist. In diesem Fall hätte man eine probabilistische Analyse eines deterministischen Algorithmus bezüglich der angenommene ~~Eingabe~~ Verteilung der Eingabe.

→ Man sollte aber eine solche Annahme über die (uniforme oder andere) Verteilung der Eingabe kritisch betrachten, und zumindest durch Stichproben experimentell nachweisen.]