UPT, Anul I CTI 2015

Proiect P&S, Tema 2

Proiectarea cache-ului pentru conţinut internet

Distribuţia Zipf este o distribuţie discretă ce modelează popularitatea conţinutului internet. În general distribuţiile internetului (Zipf şi Pareto sunt distribuţii cu coadă lungă *long tail distributions* sau *heavy tail ditributions*).

Ce înseamnă această coadă? Dacă X este o variabilă aleatoare cu coadă lungă şi x o valoare pozitivă foarte mare, $P(X > x)$ deşi este neglijabilă, este nenulă. Cu alte cuvinte variabilele cu coadă lungă iau valori foarte mari, dar cu probabilitate mică (vezi imaginile din Notebook-ul inclus).

Distribuţia de probabilitate este în procent foarte mare repartizată valorilor mici ale variabilelor, iar restul valorilor foarte mari.

În acest proiect studiaţi distribuţia Zipf în următorul context:

Presupunem că lucrezi la startup-ul *Viaţă veselă*, ce pune la dispoziţie conţinut generat de utilizatori `http://en.wikipedia.org/wiki/User-generated_content`.

Printre altele utilizatorii pot să posteze filme şi în acelşi timp să le urmărească online. Ţi se dă task-ul de proiectare a cache-lui pt cererea celor $n = 200$ de filme de care dispune până în prezent *Viaţă veselă*. Filmele sunt codificate $1, 2, \ldots 200$. Într-un fişier `DateFilme.txt` sunt înregistrate codurile filmelor urmărite online într-o perioadă de timp.

Pentru a decide popularitatea filmelor şi a salva în cache filmele ce sunt cerute/vizionate de 75% dintre utilizatori, trebuie să identifici din date distribuţia Zipf ce aproximează distribuţia de de probabilitate a popularităţii filmelor.

Distribuţia de probabilitate a popularităţii filmelor, "se învaţă" din datele din fişier astfel:

- Citeşti datele din fişier într-un vector `fv` (filme vizionate). Fie L lungimea vectorului.
- Defineşti tipul de date `Film` astfel:

```
typedef struct Film{
 int cod; // codul filmului
 int rank;// popularitatea filmului
 int nrv;// nr de vizionari ale filmului avand codul cod
 double pre;// probabilitatea de vizionare a filmului
 }Film;
```

şi un pointer la `Film`:

```
Film *film;
```

Aloci memorie pt n=200 de filme:

```
film=malloc(n*sizeof(Film));
```

La început se iniţializează doar membrul cod, şi anume: `film[k].cod=k+1`, $k = 0, 1, \ldots, n-1$.

`film[k].rank` reprezintă rank-ul filmului având codul $k+1$, în ierarhizarea filmelor în ordinea descrescătoare a popularităţii lor, `film[k].nrv`, numărul de vizionări şi respectiv `film[k].pre` este probabilitatea experimentală de vizionare a filmului cu codul $k+1$.

• Determinaţi din analiza coordonatelor vectorului `fv` nr de vizionări ale fiecărui film. Cum? Într-un caz simplu, daca vectorul `fv` are cooordonatele:

$$fv = [6, 3, 2, 7, 6, 1, 3, 3, 4, 3, 2, 6, 5, 6]$$

înseamna că în perioada monitorizată au fost vizionate 7 filme, codificate 1,2,3,4,5,6,7:

- filmul 1 a fost vizionat o dată ;
- filmul 2 a fost vizionat 2 ori;
- filmul 3 a fost vizionat de 4 ori;

etc

Numărul de vizionări ale fiecărui film este calculat de o funcţie
`void, frecventa(int *fv, int L, int n, Film *film);`

care determină de câte ori a fost vizionat filmul k şi setează corespunzător `film[k].nrv`, $k = 0, 1, \ldots n-1$.

• se sortează apoi filmele in ordinea descrescătoare a numărului de vizionări.

Initial avem `film[0].cod=1, film[1].cod=2` ... `film[n-1].cod=n`

Dacă de exemplu, în urma sortării rezultă că filmul cu codul 32 este cel mai popular, apoi filmul 12, etc şi ultimul este filmul 81, atunci se setează `film[0].cod=32, film[1].cod=17`, ... `film[n-1].cod=81` (adică se resetează membrul `cod` pt fiecare film);

În cursul sortării se setează şi rank-ul fiecărui film:

`film[0].rank=1, film[1].rank=2`, ... `film[n-1].rank=n`

Deci ATENŢIE cum implementaţi sortarea!!!

• Calculaţi apoi `Nrtv`, numărul total de vizionări ale celor n de filme, adică: `Nrtv` = $\sum_{k=0}^{n-1}$ `film[k].nrv`. Ea trebuie să fie egală cu nr L al inregistrărilor din fişierul `DateFilme.txt`

• Probabilitatea experimentală de vizionare a fiecărui film din cele $n = 200$ se calculeaza ca fiind $film[k].pre = \dfrac{film[k].nrv}{Nrtv}$, $k = 0, 1, \ldots n-1$.

După aceste calcule am definit practic o variabilă aleatoare ce ia ca valori popularitatea filmelor, şi probabilităţile sunt probabilităţile deduse din frecvenţele de vizionare:

$$X = \begin{pmatrix} 1 & 2 & \ldots & n \\ film[0].pre & film[1].pre & \ldots & film[n-1].pre \end{pmatrix}$$

Dorim să studiem dacă această variabilă aleatoare este aproximată suficient de bine de o v.a Zipf:

$$Z = \begin{pmatrix} 1 & 2 & \ldots & n \\ \dfrac{c}{1^\alpha} & \dfrac{c}{2^\alpha} & \ldots & \dfrac{c}{n^\alpha} \end{pmatrix}$$

Teoretic distribuţia variabilei Z, adică $p_k = P(Z = k)$, se exprimă astfel $p_k = \frac{c}{k^\alpha}$, $k = 0, 1, \ldots n-1$.

Logaritmând avem:

$$\ln(p_k) = \ln(c) - \alpha \ln k$$

Inmulţind cu -1 rezultă că $-\ln(p_k) = \alpha \ln(k) - \ln(c)$ şi notând $x = \ln k, y = -\ln(p_k)$, $b = -\ln(c)$, relaţia devine:

$$y = \alpha x - b$$

adică pentru orice $k = 0, 1, \ldots n-1$, punctele de coordonate $(\ln(k), -\ln(p_k))$ verifică ecuaţia unei drepte de pantă α.

Distribuţia variabilei X, dedusă din date este aproximativ o distribuţie Zipf dacă punctele asociate $(\ln(k), -\ln(film[k].pre)$ sunt uşor împrâştiate în jurul unei drepte. Cum n=200 puncte obţinute din date experimentale este puţin probabil să fie coliniare, determinăm dreapta celor mai mici pătrate asociate acestor puncte şi analizăm (vizual şi computaţional) cât de bine se potriveşte modelul liniar, logaritmilor datelor.

Dacă punctele sunt suficient de apropiate de dreapta celor mai mici pătrate, atunci panta dreptei va fi parametrul α al distribuţiei Zipf aproximative a datelor.

Pentru a genera şi analiza dreapta celor mai mici pătrate procedati astfel:

• Scrieţi într-un fişier `NumeLeader2.txt`, pe câte o linie coordonatele celor $n = 200$ puncte:

ln(`film[k].rank`) $-$ ln(`film[k].pre`)

Atenţie sunt 2 nr pe o linie, nu cumva să interpretaţi semnul minus ca diferenţă!!!!!!!!!!!!!

• Citiţi acest fişier în IPython Notebook-ul `NumeLeaderPr2.ipynb`, desenaţi punctele, calculaţi dreapta celor mai mici pătrate, deci coeficienţii ce definesc dreapta şi un parametru r, numit coeficientul de corelaţie (îl studiem la curs în săptămâna 11-16 mai). Dacă coeficientul $r > 0.9$ considerăm că modelul liniar este potrivit pentru date şi deci panta dreptei, α, dă parametrul Zipf al ditribuţiei aproximative a datelor din fişierul `DateFilme.txt`.

Introduceţi apoi în codul `C/C++` valoarea pentru parametrul α ce caracterizează distribuţia Zipf aproximativă a datelor

• Definiţi o funcţie `double *Zipf(double alpha, int n)` ce returnează vectorul probabilităţilor $pr[k]$, $k0, 1, 2 \ldots, n-1$ asociate distribuţiei Zipf de parametru α (luaţi doar 2 zecimale pt α).

• Salvaţi într-un fişier `ProbsExp.txt` probabilităţile experimentale `film[k].pre` $k = 0, \ldots n-1$ şi într-un alt fişier `ProbsTheor.txt` aceste probabilităţi ale distribuţie teoretice ce aproximează distribuţia datelor; (la sfârşit vor fi vizualizate într-un notebook cele două distribuţii)

• Calculati coordonatele vectorului F, $F[0] = 0$, $F[1] = pr[0]$, $F[2] = pr[0] + pr[1]$, ..., $F[n] = pr[0] + pr[1] + \cdots + pr[n-1] = 1$

• Deduceţi prin calcul până la ce rang de popularitate k trebuie salvate cópii ale fimelor în cache, pt a putea răspunde rapid cererilor de vizionare a filmelor din top 25%. Adică determinaţi cel mai mic k pentru care $P(X \leq k) \geq 0.75$.

• Pentru a testa că alegerea filmelor pentru cache este OK, simulaţi variabila aleatoare Zipf de parametru α, generând 2000 de valori sau mai mult (adică cereri de vizionare de filme) şi număraţi ce procent din valorile generate sunt coduri ale filmelor salvate în cache. Afişati acest procent cu un mesaj, pt a întelege ce reprezintă. Ce părere aveţi despre procentul găsit? Ce ilustrează el?

Pentru simularea acestei variabile aleatoare discrete, nu se aplică algoritmul din curs pentru că variabila ia valori n=200 valori şi ar fi ineficient. Şi anume se foloseţe vectorul F definit mai sus, în care s-au precalculat sume ale probabilităţilor consecutive (ATENŢIE, spre deosebire de algoritmul din curs aici $F[0] = 0$; adică îl includem şi pe 0 în vectorul F). Algoritmul de simulare foloseşte căutarea binară pentru a determina intervalul $(F[k-1], F[k]]$ în care cade numărul `u = urand()`:

```
1: function SimulZipf(n, F)
2:     u=urand();
3:     L=0; R=n;//L=Left, R=Right
4:     while (L<R){
5:     m=(L+R)/2;
6:     if (u<F[m]) R=m;
7:     else L=m+1;
8:     }
9:     return L;//u ∈ (F[L − 1], F[L]);s-a produs un ev de prob pr[L-1]
10: end function
```

Atentie! Notebook-ul inclus se ruleaza de 2 ori. Prima data doar pana ce afiseaza dreapta celor mai mici patrate si paramatrul α pentru distribuţia Zipf aproximantă.

A doua oară rulaţi din nou de la început până la sfârşit şi veţi obţine vizualizarea comparativă a celor două distribuţii: distribuţia datelor şi distribuţia Zipf aproximantă.

Atât prima oară, cât şi a doua se generează câte o imagine pe care apoi o includeţi in arhiva cu această temă de proiect.