Proiectul nr 3, P&S

Lanţuri Markov.
Simularea navigării aleatoare pe WEB şi simularea unui protocol de rutare

Această tema de proiect are 2 părţi. Prima parte este fundamentată în Cursul 14, iar a doua în Cursul 15.

Implementaţi cele două părţi folosind un `switch` cu două cazuri. NU cereţi utilizatorului (adică mie) să introducă de la tastatură cazul pe care vrea sa-l testeze. Setaţi voi iniţial un caz şi apoi eu când verific, schimb în cod cazul şi recompilez!!!!!

Pentru prima parte aveţi nevoie de următoarele date:

• Matricea hyperlink, H, asociată unui graf WEB cu m noduri, cu m cuprins între 15 şi 20.

Elementele matricii H le citiţi dintr-un fişier `hyperlink.in`

• Din cele m pagini, cel puţin 3 trebuie să fie pagini *dangling*.

• definiţi structura de date:

```
typedef  struct Page{
         int inlinks;//numarul de inlinkuri  spre nod
         int outlinks;// numarul de outlinkuri din nod
         int nrvis;// numarul de vizite ale paginii
         double pgrkE;//pagerankul experimental
         double pgrkA;//pagerankul paginii calculat cf algoritmului }Page;
```

• declaraţi un pointer la `Page` astfel:

`Page *pag;`

şi alocaţi memorie

• După ce s-a alocat memorie pentru `pag`, `pag[i]` este structura asociată paginii i, i=0,1,2, ... m-1;

• `pag[i].inlinks` reprezintă numărul de inlinkuri ale paginii cu codul i, etc.

• După ce citiţi matricea hyperlink din fişier, calculaţi pentru fiecare pagină numărul de inlink-uri şi outlink-uri;

• Construiţi matricile intermediare care conduc la matricea Google, G, din curs (folosiţi parametrul $\alpha = 0.85$).

Toate matricile implicate se declară în felul următor:

```
double **A;
int i;
A=(double **)malloc(m*sizeof(double *));// se aloca memorie pentru m linii

/* pentru fiecare linie se aloca memorie pt m locatii
de sizeof(double) bytes  si  se initializeaza pe 0:*/

for(i=0;i<m;i++)
  A[i]=(double *)calloc(m, sizeof(double));// calloc aloca memorie si
                                           //initializeaza pe 0 fiecare A[i][j]
```

Cei care ştiu `C++` pot folosi operatorul `new`.

a) Efectuaţi simularea navigării aleatoare pe graful WEB pentru a deduce Pagerank-ul experimental al fiecărei pagini.

Pagerank-ul experimental, `pag[i].pgrkE`, al paginii i este frecvenţa asimptotică cu care un surfer aleator vizitează pagina i, adică numărul de vizite ale paginii i făcute de surferul aleator, din cei $N = 1000$ sau $N = 2000$ paşi (deci N foarte mare, pt a fi considerat ca tinzând la ∞) pe care surferul îi face navigând pe WEB.

Trecerea de la o pagină la alta se simulează, simulând o traiectorie a lanţului Markov definit de nodurile $S = \{0, 1, 2, \ldots, m-1\}$, matricea de tranziţie–matricea Google G, iar distribuţia iniţială de probabilitate arbitrară (o alegeţi voi, uniformă sau nu).

• Pentru simularea lanţului Markov aveţi nevoie de o funcţie care simulează o distribuţie discretă de probabilitate (adică ceea ce în curs am notat simbolic:

`simulator(`$0, 1, 2, \ldots,$ `m` $-$ `1; p)`

Algoritmul de simulare îl găsiţi în Cursul 12!!!

În algoritmul de simulare a lanţului Markov, la fiecare pas se actualizează vectorul probabilist `p`. Pentru a implementa linia `p` $=$ `Q[i,:];`

din pseudocodul algoritmului de simulare (Cursul 14) folosiţi aritmetica pointerilor.

Simularea trebuie să genereze o traiectorie în graful WEB de N paşi ($N = 1000$ sau $N = 3000$; experimentaţi cu diverşi N).

• Pentru a estima numărul de vizite ale fiecărei pagini, respectiv frecvenţa cu care este vizitată fiecare pagină, procedaţi astfel:

După fiecare vizită a unei pagini i, incrementaţi cu o unitate, `pag[i].nrvis` şi la sfârşit calculaţi `pag[i].pgrkE=(double)pag[i].nrvis/N` (acesta este PageRank-ul experimental sau frecvenţa cu care a fost vizitată pagina i).

b) Calculaţi apoi vectorul PageRank algoritmic, adică vectorul

$\pi = [\pi[0], \pi[1], \ldots, \pi[$`m` $-$ `1`$]] =$ [`pag[0].pgrkA, pag[1].pgrkA,` $\ldots,$ `pag[m` $-$ `1].pgrkA`]

aplicând metoda puterii şi setând ϵ ca fiind `eps=1.0e-05`, iar distribuţia iniţială de probabilitate, una pe care o alegeţi voi (atenţie suma coordonatelor (pozitive) ale distribuţiei iniţiale de probabilitate trebuie să fie 1!!!!!).

• Calculaţi în funcţia ce implementează metoda puterii şi numărul de iteraţii, `nrIt`, necesare până la convergenţa metodei puterii.

• Ordonaţi cele m pagini în ordinea descrescătoare a Pagerank-ului lor algoritmic şi le afişati pe ecran, într-un tabel ce are coloanele:

`Pagina, Nr inlinkuri, Nr outlinkuri, PageRankA, PageRankE`

• Afişaţi şi numărul de iteraţii `nrIt`, în care s-a parcurs bucla din metoda puterii.

Testaţi pentru mai multe distribuţii initiale de probabilitate şi în Concluziile temei de proiect precizaţi dacă aţi obţinut acelaşi PageRank algoritmic pentru fiecare sau nu!!!

2. În această parte determinaţi performanţa unui protocol de forwardare a pachetelor de informaţie, într-o reţea wireless, după ce s-a atribuit traiectoria de la sursă la destinaţie. Protocolul de rutare între sursă şi destinaţie este definit de un lanţ Markov absorbant ilustrat în figura de mai jos, pentru cazul în care destinaţia este cel de-al 5-lea nod:

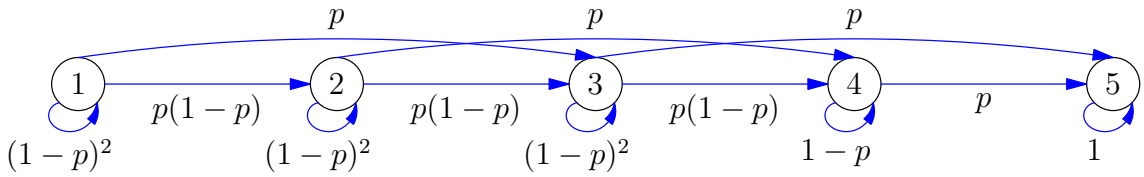

Fig. 1: Protocolul de forwardare.

O măsura a performanţei (teoretice) a protocolului este timpul mediu petrecut de un pachet în reţea, înainte de a ajunge la destinaţie. Din Cursul 15 se ştie că timpul mediu petrecut de un pachet ce porneşte din nodul 1, în stările tranzitorii (deci în reţea), înainte de a fi absorbit (adică înainte de a ajunge la destinaţie) este egal cu suma elementelor de pe prima linie a matricii fundamentale $N = (I - T)^{-1}$.

Performanţa experimentală se evaluează simulând mai multe forwardări (de exemplu `nrf=1000`) şi dupa fiecare calculand numarul, `nrnod`, de noduri disticte sau nu, vizitate de pachet înainte de a ajunge la destinaţie. După cele `nrf` forwardari simulate, se calculeaza media aritmetică a numarului de noduri vizitate.

Fiecare nod este vizitat într-o unitate de timp şi deci aceasta medie este timpul mediu experimental petrecut în reţea.

Simularea unei forwardări înseamnă simularea lanţului Markov având m noduri şi distribuţia iniţială de probabilitate, $\pi 0 = [1, 0, 0 \ldots, 0]$, adică cu probabilitatea 1, pachetul porneşte din nodul sursă 1. Simularea unei forwardări se încheie când pachetul a ajuns la destinaţie.

• Consideraţi o reţea de cel puţin m=20 noduri şi definiţi matricea de tranziţie, nu setând manual fiecare element, ci observând din figura dată care este regula de setare. Mai precis definiţi o funcţie:

```
double **Qtrans(int m, double p);
```

În corpul funcţiei declaraţi:

`double **Q;`
alocaţi memorie şi iniţializaţi conform regulii observate în figură.
Funcţia are ultima linie:
`return Q;`// scuze pt cei pt care era evident :)

După ce aţi definit şi iniţializat matricea de tranziţie, o salvaţi într-un fişier `QtransP.txt`, ce va fi citit în Notebook-ul `NumeLeader3.ipynb` în care se calculează matricea fundamentală şi timpul mediu teoretic petrecut în reţea. **Litera P din numele fşierului** se inlocuieste cu `06` sau `08, 085`, dupa cum a fost setată probabilitatea p din definiţia matricii de tranzitie Q, adică, $p = 0.6, p = .8, p = 0.85$ (recurgem la acest artificiu nefiresc pt a fi cât mai simplu de testat in Notebook).

Atenţie!!! în fişierul `QtransP.txt` scrieţi linie dupa linie elementele matricii, nu şi dimensiunea matricii, pt ca în `Python` aflăm prin interogare care este `shape`-ul, adică nr linii şi coloane!!!!

Pentru teste folosiţi probabilitatea $p = 0.6$, $p = 0.85$ şi precizaţi în Concluziile temei de proiect, comparativ, performantă teoretică si experimentală.

Pe ecran afişaţi nr de noduri ale reţelei, probabilitatea p setată şi timpul mediu `tmedT` teoretic calculat în Notebook şi `tmedE`, timpul mediu experimental, rezultat din simularea celor `nrf` forwardări.