СЛУЧАЙНО СЪРФИРАНЕ В ИНТЕРНЕТ

Евгения Стоименова

Резюме. Целта на тази статия е да обясним една от основните идеи за оценка на посещаемостта на web страниците. Ще използваме основни програмни средства на езика C++, за да изучим модела на случайното сърфиране. Материалът е достъпен за ученици от средния курс, с базови знания по информатика и езика C++.

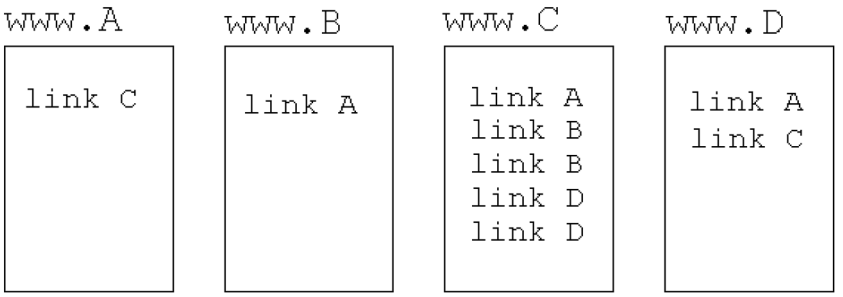
*Keywords***:** random surfer, web page ranking, stochastic simulation

Един от най-атрактивните алгоритми за определянето на важността на страниците от интернет мрежата World Wide Web е предложен от Сергей Брин и Лари Пейдж през 1998 г. Алгоритъмът е реализиран в Google, компанията, която те създават. Търсачката на Google подрежда страниците от зададен критерий съобразно тяхната важност (обикновено това са страниците, най-близки до критерия на търсене и най-полезни за потребителя). Естествено е страниците с високи честоти на посещаемост да излизат първи при търсене, защото те се очакват от много повече потребители.

Откъде Google знае честотите на посещаемост на всички web страници? *Случайно сърфиране* е идеализиран модел на поведение на потребителите на Web, които преминават от една страница към друга донякъде случайно. При това движение се отчитат посещенията в различните страници от мрежата. При дълго сърфиране честотите на посещение се стабилизират и са еквивалентни на ранговете на страници PageRank, които използва Google. Подходът за изложение на идеята в тази статия до голяма степен е взаимстван от [3]. За разбирането и не са необходими някакви специални математически знания, освен понятията вектор и матрица, които учениците знаят от часовете по информатика.

Мрежата. Интернет мрежата се разглежда като множество от страници, които са свързани помежду си чрез (хипер)връзки. Към всяка страница са насочени връзки от други страници и от всяка страница излизат връзки към други страници от мрежата.

Интернет потребителят преминава от една страница към друга като избира връзка от списъка в страницата, в която се намира, или като изпише името на страницата в съответното поле на браузера. При това движение потребителят



Фигура 1 Web страници

обхожда мрежата и попада в различни страници. Това обикновено се нарича сърфиране из интернет.

Случайно сърфиране. При случайното сърфиране се предполага, че интернет потребителят избира връзка от списъка на страницата, в която се намира, по *случаен* начин. Всички връзки се смятат за равновероятни. При много дълго случайно сърфиране могат да се отчетат честотите на посещение в различните страници. Това може да е индикатор за важността на отделните страници, а резултатът е еквивалентен на ранговете на страниците PageRank, въведени от Брин и Пейдж и реализиран в търсачката на Google.

Моделът изглежда съвсем прост, но има някои особености. В интернет мрежата съществуват много страници, от които няма изходящи връзки. Има и такива страници, които създават затворен кръг, от които също не може да се излезе. За да се избегнат тези „дупки", в модела се допуска от време на време да се прави скок към произволна страница от мрежата. Ето формалното описание на модела на случайното сърфиране.

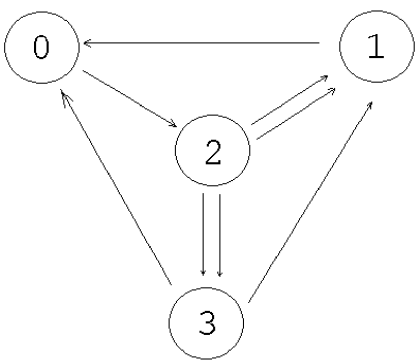
> Случайното сърфиране по правилото 90-10 се определят по следния начин*: в 90% от случаите потребителят избира случайно връзка от страницата, в която се намира. В останалите 10% от случаите той преминава към случайно избрана страница от мрежата като по този преход всички страници от мрежата се смятат за равновероятни.*

Случайното сърфиране за конкретна мрежа може да се симулира и да се отчетат честотите на посещение на отделните страници. Тези честоти ще определят рангове на страниците.

Очевидно, поведението на реалния потребител не е толкова просто. Никой не избира връзките или страниците случайно, а правилото 90-10 (или друго зададено) е само хипотетично. Няма реална възможност да се премине директно към произволна страница от мрежата и обикновено са достъпни част от web страниците. Въпреки тези слабости, моделът на случайното сърфиране е достатъчно добър, за да се изследват свойствата на web мрежата.

Пример. Да построим модел на мрежа, състояща се от страниците на предишната фигура.

Мрежата има четири страници и общо девет връзки между тях. Моделът е представен с граф. Възлите на графа представляват страниците, (номерирането не е от значение и тук е съответно на реда A-B-C-D). Всяка стрелка означава една връзка от страница към страница. Движението (сърфирането) се извършва от един връх до друг само ако има стрелка в тази посока.



Фигура 2 Схема на мрежа

Да започнем с въпроса: Коя страница от фигурата се посещава най-често при случайно сърфиране? Страница 1, 3, 0 или може би 2? Може да се даде интуитивен отговор на този въпрос, а след това да го сравним с резултата от симулирано движение по мрежата.

Входни данни. Форматът на входните данни ще организираме така, че да можем да задаваме информацията за различни мрежи, не само за примера, който имаме. Така ще имаме възможност да изучаваме случайното сърфиране в различни мрежи. Данните ще запишем в масива `inpdata[]`.

Данните е добре да се прочетат от файл като последователни числа, разделени с интервал. Първото число трябва да е броя на страниците. Връзката от една страница до друга представяме с наредена двойка от числа, първото съответно на страницата, която задава връзката, а второто - страницата, към която е насочена.

```
4
0 2
1 0
2 0  2 1  2 1  2 3  2 3
3 0  3 1
```

Фигура 3 Вход за програмата

Да отбележим, че при поточно

четене на данни, един или повече интервала се приемат за един, нов ред също е равносилен на интервал. Затова данните могат да се оформят така, че да са удобни за визуално възприемане. В примера, първото число 4 е за броя на страниците, в следващия ред е записана единствената връзка, която има от стр. 0 към стр. 3 и т.н.

Бележка: За нуждите на тази задача, може данните да зададем чрез присвояване, вместо да ги четем от файл:

```
int inpdata[] ={4,0,2,1,0,2,0,2,1,2,1,2,3,2,3,3,0,3,1}
```

Преходна матрица. Движението на потребителя, извършващ случайно сърфиране в мрежата, ще опишем чрез така наречената *матрица на преходните вероятности*. Ако страниците са *N*, то матрицата е с размер *N x N* и в нея елементът на *i*-тия ред и *j*-тия стълб съдържа вероятността за преход от страница *i* в страница *j*. Първата задача е на напишем функция, която да създава тази матрица при зададен вход. Ще приложим, освен това, и правилото 90-10. Задачата не е трудна и ще я опишем в три стъпки:

- Четем `N` и създаваме 3 масива:

`btwnlnks[N][N]}` - за броя връзките между всеки две страници;
`outlnks[N]`- за броя на връзките, излизащи от всяка страница; и
`P[N][N]` - за преходната матрица.

- Четем връзките от входа и натрупваме:

в `btwnlnks[i][j]}` - броя на връзките от страница *i* до страница *j*;
в outlnks[i]} - броя на връзките, излизащи от страница *i*.

- Прилагаме правилото 90-10, за да изчислим елементите на преходната матрица `P[][]`.

Първите две стъпки са елементарни, третата става по следния начин: умножаваме `btwnlnks[i][j]` с 0.90/`outlnks[i]` ако има връзка от *i* до *j* (избор на връзка от списъка с вероятност 0.9), и прибавяме 0.10/`N` към всеки елемент (преход към случайна страница от мрежата с вероятност 0.1).

За данните от примера двумерният масив `btwnlnks[][]`, представен с матрица, изглежда по следния начин:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 2 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Масивът `outlnks[]` е едномерен и има вида [1 1 5 2].

За да получим масива `P[][]`, трябва първо да умножим елементите на `btwnlnks[][]` с 0.9. След това елементите от всеки ред, *i*, да разделим на съответния елемент `outlnks[i]`. Накрая към всеки елемент от масива да добавим 0.025. (Числото 0.02 се получава като 0.1 от правилото 90-10 се раздели по равно между четирите страници.) Това, записано чрез матрици, изглежда така:

$$\begin{bmatrix} 0 & 0 & 0,9 & 0 \\ 0,9 & 0 & 0 & 0 \\ 0,18 & 0,36 & 0 & 0,36 \\ 0,45 & 0,45 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0,025 & 0,025 & 0,025 & 0,025 \\ 0,025 & 0,025 & 0,025 & 0,025 \\ 0,025 & 0,025 & 0,025 & 0,025 \\ 0,025 & 0,025 & 0,025 & 0,025 \end{bmatrix}$$

Така преходната матрица `P[][]` става:

$$= \begin{bmatrix} 0,025 & 0,025 & 0,925 & 0,025 \\ 0,925 & 0,025 & 0,025 & 0,025 \\ 0,205 & 0,385 & 0,025 & 0,385 \\ 0,475 & 0,475 & 0,025 & 0,025 \end{bmatrix}.$$

Функцията `Transition` изчислява преходната матрица. Тя превръща входен масив от връзки на модела в матрица от преходни вероятности. Масивите `btwnlnks`, `outlnks` и `P` трябва да се инициализират предварително:

```
int outlnks[20]={0}; // number of links out a page
int btwnlnks[20][20]={0}; // number of links between pages
double P[20][20]={0}; // transition matrix
```

Числото 20 за максималния брой на страници е примерно, то може да се фиксира съобразно големините на мрежите, които ще се анализират.

И така, входът за `Transition` е масивът `inpdata[]`, изходът - преходната матрица `P[][N]`. Ето и кода на функцията:

```
void Transition( int arr[], int btwnlnks[][20], int outlnks[20],
  double P[][20] )
```

```
{
int N=arr[0]; // number of pages
int n=sizeof(arr)/sizeof(arr[0])-1; // number of links
for (int k=1; k<n; k +=2) {
int i = arr[k];
int j = arr[k+1];
outlnks[i]++;
btwnlnks[i][j]++;
}
for (int i=0; i<N; i++){// Calculate probability for column j.
for (int j = 0; j < N; j++)
P[i][j] = .90*btwnlnks[i][j]/outlnks[i] + .10/N;
}
}
```

В преходната матрица всички числа са различни от нула, което означава, че всички страници са достижими за един преход. Числата по диагонала са вероятностите за оставане в същото състояние за един ход. Не е ограничение, това че правилото 90-10 допуска такава възможност.

Преходната матрица има и други интересни свойства, които са съществени за задачата, която сме си поставили. Едно от тях е, че всеки ред (напр. i-тия ред) на матрицата съдържа вероятностите за преход от съответната страница (i-тата) до всяка друга страница за един ход. Другото, което ще използваме е, че сумата на числата по редове е 1.

Симулация. Целта на симулираното движение по мрежата е да получим поредица от страници (номера), които да са посетени при случайното сърфиране. Този изход може да се изведе на печат или визуализира върху графа, но по-съществено е да се преброят колко пъти се среща всяка страница в тази поредица.

За симулацията имаме нужда от преходна матрица и начална страница. Симулацията ще направим с две функции - `NextPage` и `Surfing`. Входните аргументи на `NextPage` са текущата страница и преходната матрица, а изхода е следващата страница за един ход съгласно правилата на случайното сърфиране. `Surfing`} прави зададен брой (`T`) прехода чрез `NextPage` и отчита броя на посещение за различните страници. Този брой се разделя на `T` и се получават честотите на посещение. При много голямо `T` честотите ще са много близки до ранговете на страниците PageRank.

Преход за една стъпка (`NextPage`). Основната стъпка при симулацията на случайното сърфиране е преходът от една страница към друга за един ход.

Променливата `Page` съдържа номера на текущата страница. Редът от преходната матрица `P[Page]`, съдържа преходните вероятности от страница `Page` до всичките `N` страници от мрежата. Тези вероятности трябва да използваме, за да определим следващата страница.

С други думи, когато знаем в коя страница е потребителя, задачата е да се генерира случайно цяло число (страница) между 0 и `N-1`, съгласно зададените вероятности за тази страница. Ето как може да стане това.

Използваме функцията `rand()` за генериране на случайно число и трансформацията `rand()/RAND_MAX`, за да получим реално число `r` в интервала [0,1]. Това число ще използваме, за да изберем следващата страница след `Page`.

Първо разделяме мислено интервала [0,1] на `N` подинтервала с дължини съответни на големините на вероятностите, от реда `P[Page]` на преходната матрица. Точките на деление поставяме последователно така: Първата точка след 0 е `P[Page][0]`, следващите са

`P[Page][0]+P[Page][1]}, P[Page][0]+P[Page][1]+P[Page][2],` и т.н.

Всяка точка получаваме от предишната с добавяне на поредната вероятност от реда `P[Page]}`. Тези точки определят `N` подинтервала на интервала [0,1]. На всеки от тях съпоставяме една страница.

Сега трябва да проверим в кой подинтервал попада генерираното случайно число `r` и да изберем страницата, която съответства на този подинтервал. Резултатът е следващата страница след `Page`. Вероятността `r` да принадлежи на всеки от подинтервалите е пропорционална на дължините на тези подинтервали и съответно изборът на страница съответства на желаните вероятности.

Това става с цикъл `for` по следния начин: Променливата `sum` последователно приема за стойности точките на деление на интервала, проверява дали случайното число `r` е преди точката в `sum` и прекъсва цикъла, когато я надхвърли. Кодът на функцията изглежда така:

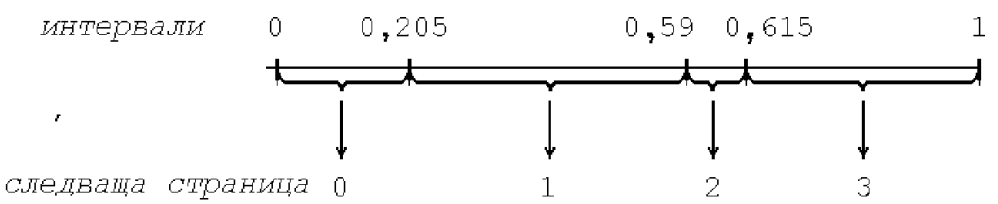
```
void NextPage( int page, double P[20][20], int N )
{
double r = rand()/RAND_MAX;
double sum =0.0;
for (int j=0; j<N; j++)
{
sum +=P[page][j];
if (r<sum) { page = j; break;}
```

```
    }
    return page;
    }
```

Да разгледаме пример за действието на функцията `NextPage`. Нека потребителят се намира в страница 2. Преходните вероятности от преходната матрица са 0,205; 0,385; 0,025 и 0,385. Променливата `sum` последователно приема стойностите 0, 0,205; 0,59; 0,615 и 1. С тези стойности са определени интервалите:

[0, 0,205], [0,205; 0,59], [0,59, 0.615]; [0,615; 1]

- по един за всяка страница. Получаваме делението, изобразено на фигурата. На всеки интервал съответства номер на страница.



Да предположим, че чрез функцията за случайно число сме получили числото 0,43. Цикълът се върти по *j* от 0 до 1 и прекъсва там, тъй като числото 0,43 е в втория интервал (0,025; 0,5). Така следващата страница за потребителя е 1.

Движение по мрежата. Движението по мрежата чрез случайно сърфиране има за цел да се отчетат честотите на посещение в различните страници. При фиксиран брой прехода (напр. 1000), общият брой преходи се разпределя между страниците и се получават относителните честоти на посещение. Ако относителните честоти се разделят на общия брой (в случая 1000), се получават *абсолютните честоти* на посещение. Така тези честоти не са свързани повече с общия брой преходи, *сумата им е 1*, и се интерпретират по един и същ начин за различно по време сърфиране.

Функцията `Surfing` повтаря `NextPage T` на брой пъти. В масива `freq[]` се записва броя на посещенията във всяка страница. След `T` хода, елементът `freq[Page]` на масива ще съдържа честотата на поява на страницата `Page`. Масивът `freq` трябва да се инициализира преди изпълнението на `Surfing`

```
int freq[100]={0}; // page frequencies
```

Кодът на функцията изглежда така:

```
 void Surfing(int page, double P[20][20],int N, int T, int freq[20])
 {
 for (int j=0; j<T; j++)
 {
 NextPage(p,page,N);
 freq[page]++; // Да се раздели на T някъде !!!!
 }
 return freq;
}
```

Смисълът на симулацията с `Surfing` е да се отчита честотата на посещаемост на отделните страници. За всеки краен брой итерации, тези честоти са приблизителни. При увеличаване на броя на итерациите, се увеличава точността на честотите поради едно от свойствата на преходната матрица. (Свойствата на преходната матрица ще разгледаме специално по-нататък.)

Експеримент. Да разгледаме резултатите от няколко симулации на случайно сърфиране. Започваме примерно от страница 0 и след 1000 прехода честотите на посещение на страниците (закръглени до третия знак след десетичната запетая) са:

Начална страница	**Честоти на страниците**			
	0	1	2	3
[0]	0,335	0,198	0,323	0,144
[0]	0,333	0,196	0,326	0,145
[0]	0,339	0,194	0,328	0,139
[0]	0,320	0,198	0,327	0,155
[0]	0,332	0,214	0,320	0,134
[0]	0,338	0,199	0,322	0,141

Резултатите са подобни когато сърфирането започне от друга страница.

Начална страница	**Честоти на страниците**			
	0	1	2	3
[1]	0,335	0, 200	0, 327	0, 133
[1]	0,323	0,209	0,321	0,147
[1]	0,333	0,194	0,322	0,151
[2]	0,324	0,210	0,315	0,151
[2]	0,324	0,206	0,324	0,146
[2]	0,326	0,208	0,319	0,147
[3]	0,331	0,207	0,324	0,138
[3]	0,325	0,202	0,329	0,144
[3]	0,332	0,204	0,320	0,144

След 1000 прехода честотите на посещение на четирите страници не зависят от началната страница! Този феномен може би да изглежда странен, но има обяснение, което отново се крие в свойствата на преходната матрица.

Честотите от отделните симулации са близки, но тяхната точност не е голяма. При 1000000 итерации честотите от една симулация са:

	Честоти на страниците			
страници	0	1	2	3
честоти	0,3310829	0,2045380	0,3231330	0,1412460

По-натакък ще разгледаме и едно друго приближение на честотите, следствие от случайното сърфиране, но осигуряващо по-бърза сходимост към истинските рангове. Чрез него могат да се получат следните честоти:

	Честоти на страниците			
страници	0	1	2	3
честоти	0,3257049	0,1848188	0,3181344	0,1713418

(Всички цифри са значещи.)

Съответстват ли тези честоти с интуитивното ни предположение за най-често посещаваната страница? Можехме ли да допуснем, че страница 0 и страница 2 са с най-високи и почти равни честоти?

Рангове на страници. Голямата популярност на Google се дължи преди всичко на ефективния алгоритъм за подреждане на web страниците по важност. Този алгоритъм се нарича PageRank и е предложен от Брин и Пейдж през 1998 г.

Ранговете на PageRank се определят напълно от структурата на хипервръзките в интернет мрежата World Wide Web. Те се преизчисляват непрекъснато и не са свързани със съдържанието на страниците или с определени критерии на търсене. За всяко конкретно търсене Google намира страниците, отговарящи на критерия, и ги подрежда съобразно техния PageRank.

PageRank е свързан с връзките, които са насочени към една страница. Ясно е, че страниците, които имат повече връзки, сочещи към тях, се посещават по-често и съответно имат по-висок ранг. Освен това страници, които се цитират в много популярни страници, би трябвало също да имат по-предни позиции в списъците на търсачката. PageRank е отражение на тези две желани свойства за подреждане на резултатите от търсене. Останалото е рекурсивно изчисляване на теглата на страниците като се използва структурата на хипервръзките между страниците.

Ще дадем формалната дефиниция на PageRank, а повече подробности могат да се намерят в статиите [1] и [2].

Нека u е произволна страница. Всички страници, които имат връзка към u, допринасят за нейния ранг. Нека B_u е множеството от тези страници. Рангът на една страница v от B_u се разпределя към страница u и към другите страници, които са цитирани в v. Ако означим с N_v броят на връзките, които са в v, то приносът на v към ранга на u е $\frac{R(v)}{N_v}$.

Така рангът на страницата u се формира чрез ранговете настраниците, които имат връзки към нея:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v},$$

където сумата е по всички страници от множеството B_u на страниците, които имат връзки към u. Нормализиращият множител c осигурява общият ранг на всички страници да е константа.

В тази дефиниция са включени всички страници от мрежата. Тя съответства на случайното сърфиране, в което не се допуска случаен скок към произволна страница от мрежата.

Има няколко особености при прилагането на тази формула на практика в реалната мрежа World Wide Web. Един съществен проблем идва от това, че страниците, от които не излизат връзки, не могат да участват във формулата,

защото за тях N_u е нула. Определянето на PageRank в www става в две итерации. При първата итерация, от множеството на всички страници се изключват тези, от които не излизат връзки и се определят ранговете на останалите страници. При второто итерация участват всички страници, изчисляват се рангове на пропуснатитестраници и се коригират ранговете на останалите страници.

Друг проблем е, че в реалната мрежа съществуват групи от страници към някои от които има връзки отвън, но от никоя от тях не излизат връзки извън групата. Тези страници са нещо като капани и с опростената формула тези страници получават безкраен ранг.

Прост капан. За да се избегнат такива проблеми, формулата се коригира подобно на правилото 90-10 по следния начин. Нека *E(u)* е вектор с дължина броя на страниците в мрежата и елементи, съответни на някакви базови рангове на страниците (може и да са равни). Тогава PageRank на това множество от web страници удовлетворява уравнението

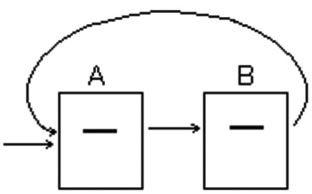


Схема на прост капан

$$R'(u) = c \sum_{v \in B_u} \frac{R'(v)}{N_v} + cE(u).$$

Ранговете на страниците са свойство на самите страници, а не на начина за изчисляването. Рангът на една страница е отражение на поведението на потребителите на WWW и е равен на вероятността тя да се бъде посетена при случайно сърфиране.

За изчисляването на PageRank не е задължително да се извършва продължително случайно сърфиране. Има по-ефективен метод за изчисляването на PageRank, който се основава на алгебричните свойства на преходната матрица и на така наречените Вериги на Марков. По-нататък ще разгледаме подробно този метод.

ЛИТЕРАТУРА

1. Brin, S., Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30, 107-117.
2. Page, L., Brin, S., Motwami, R, Winograd, T. (1998). The PageRank Citation Ranking: Bringing Order to the Web. Technical report. Stanford Digital Library Technologies Project.

3. Langville, A.N., Meyer, C.D. (2006). *Google's PageRank and beyond*, Princeton: Princeton University Press.
4. Baldi, P., Frasconi, P., Smyth, P. (2003). *Modeling the Internet and the Web. Probabilistic Methods and Algorithms*. NYSE: JohnWiley & Sons Inc.

✉ **Евгения Стоименова**
професор, доктор на математическите науки
Институт по математика и информатика
Българска академия на науките, София
http://www.math.bas.bg/~jeni/

RANDOM INTERNET SURFING

Abstract. In this paper we describe the Random Surfer Model for evaluating the importance of web pages. Google computes PageRank of a particular page from the theoretical probability of visiting that page when clicking on links at random. We assume that the reader has some basic programming knowledge in C++ to follow and reproduce the idea of this model.

✉ **Eugenia Stoimenova**
Professor, DSc in Mathematics
Institute of Mathematics and Informatics – BAS
Acad. G. Bonchev Street, bl. 8
1113 Sofia, Bulgaria
http://www.math.bas.bg/~jeni/