CS 137 Part 4

Structures and Page Rank Algorithm

October 6th, 2017

Structures

- Structures are a compound data type.
- They give us a way to group variables.
- They consist of named member variables and are stored together in memory.

Example

For example, suppose we wanted to model a clock; we would want the hours and minutes. Let's call this `tod` for time of day.

```
struct tod{
  int hours;
  int minutes;
};
```

We could then use our struct to create instances of the time. For example

- `struct tod now = {16,50};`
- `struct tod later = {.hours = 18};`

Full Code

```
#include <stdio.h>
struct tod{
  int hours;
  int minutes;
};
void todPrint(struct tod when) {
  printf("%0.2d:%0.2d\n",
      when.hours, when.minutes);
}
int main(void){
  struct tod now = {16,50};
  struct tod later = {.hours = 18};
  later.minutes = 1;
  todPrint(later);
  return 0;
}
```

More on Structures

- We can even return structures as well in much the same way as you would expect.
- Example: `struct tod todAddTime(struct tod when, int hours, int minutes)`
- Try to code this example. What issues arise from doing the naive thing?

Example

```
struct tod todAddTime(struct tod when,
   int hours, int minutes) {
  when.minutes += minutes;
  when.hours += hours + when.minutes / 60;
  when.minutes %= 60;
  when.hours %= 24;
  return when;
}
int main(void) {
  struct tod now = {16, 50};
  now = todAddTime(now, 1, 10);
  todPrint(now);
  return 0;
}
```

Reminder

- When passing structs to functions, these values are also passed by value.
- If you wanted to modify the original struct in memory, you would need to pass a pointer to it and then modify the contents of the pointers (more on this later).

Simplification

To make life easier, we can use a typedef to help create structures

```
#include <stdio.h>
typedef struct{
  int hours;
  int minutes;
} tod;
int main(void) {
  tod now = {14,40}; //instead of struct tod
  return 0;
}
```

Example

Build a Video Game Character struct. Call this `vgchar`. It should have an identification number, `x` and `y` positions starting at the origin, a power level and a defense level. Then, create some of the following functions:

- Move up, down, left, right
- Fight (between two characters, take the power levels subtract the opponents defense levels and the higher wins)
- Change ID which takes in a new integer ID number.

Page Rank

- How does Google's Searching Algorithm Work?
- Main idea: It crawls the web, indexes words on each page and then uses the index to find matches and sort by how relevant it is.
- Algorithm is due to Sergey Brin and Larry Page in their paper "The Anatomy of a Large-Scale Hypertextual Web Search Engine"
- See `http://infolab.stanford.edu/~backrub/google.html`
- Major idea: Pages with lots of links to them should be classified as good (otherwise why do they have lots of links?)

Definition of the Rank of a Page Version 1

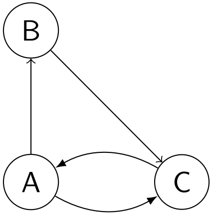
- For a page P, let Q_i be all of the pages that link to P for $1 \leq i \leq M$ with M a nonnegative integer.
- Then, the rank of a page P, denoted by $r(P)$ is

$$r(P) = \sum_{i=1}^{M} \frac{r(Q_i)}{|Q_i|}$$

  where $|Q_i|$ is the number of outbound links on page Q_i.
- We will also normalize throughout so that their sum is one.

Simple Example

Suppose the internet consisted of three pages, A, B and C. Below, the arrows indicate which pages go to which other pages.

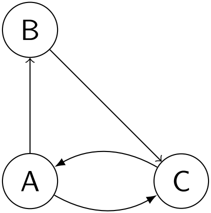


Using the formula on the previous page, we have

$$r(A) = \frac{r(C)}{1} \qquad r(B) = \frac{r(A)}{2} \qquad r(C) = \frac{r(A)}{2} + \frac{r(B)}{1}$$

Simple Example

Suppose the internet consisted of three pages, A, B and C. Below, the arrows indicate which pages go to which other pages.



Using the formula on the previous page, we have

$$r(A) = \frac{r(C)}{1} \qquad r(B) = \frac{r(A)}{2} \qquad r(C) = \frac{r(A)}{2} + \frac{r(B)}{1}$$

Solving gives $r(A) = 2r(B) = r(C)$. Normalizing so that $r(A) + r(B) + r(C) = 1$ gives $r(A) = 0.4$, $r(B) = 0.2$, $r(C) = 0.4$

Random Surfer Model

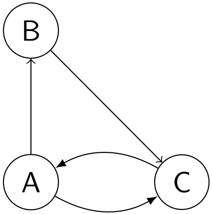
- What Google does is slightly more sophisticated than this.
- Suppose we had a web surfer that:
  - with probability δ goes to a link from the current page
  - with probability $1 - \delta$ goes to a random page
- Define the randomized page rank of a page $r_\delta(P)$ to be

$$r_\delta(P) = \frac{1-\delta}{N} + \delta \sum_{i=1}^{M} \frac{r(Q_i)}{|Q_i|}$$

where N is the number of pages on the web.

Simple Example

With this model, let's revisit our example with $\delta = 0.8$.



Using the formula on the previous page, we have

$$r(A) = \frac{1}{15} + \frac{4r(C)}{5} \qquad r(B) = \frac{1}{15} + \frac{2r(A)}{5}$$
$$r(C) = \frac{1}{15} + \frac{2r(A)}{5} + \frac{4r(B)}{5}$$

Solving

- With a little patience and the usual normalization, we can solve this system to see that

$$r(A) = \frac{61}{159} \approx 0.38365\ldots \qquad r(B) = \frac{35}{159} \approx 0.22013$$
$$r(C) = \frac{21}{53} \approx 0.39623$$

  via say Gaussian Elimination or even Cramer's Rule.
- However, if N is large... say $N > 10^9$ this is very infeasible.
- A trick that is often used is to use a fixed point iteration process called Jacobi's Method!

Jacobi's Method

- Begin with a solution $(r_0(A), r_0(B), r_0(C)) = (1/3, 1/3, 1/3)$.
- Compute

$$
\begin{aligned}
r_1(A) &= \frac{1}{15} + \frac{4r_0(C)}{5} \\
r_1(B) &= \frac{1}{15} + \frac{2r_0(A)}{5} \\
r_1(C) &= \frac{1}{15} + \frac{2r_0(A)}{5} + \frac{4r_0(B)}{5}
\end{aligned}
$$

Repeat with $(r_1(A), r_1(B), r_1(C))$ until you're happy!

For Our Example

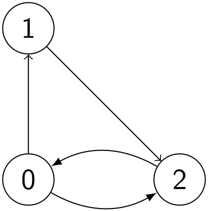
Iteration	r(A)	r(B)	r(C)
0	0.33333	0.33333	0.33333
1	0.33333	0.20000	0.46667
2	0.44000	0.20000	0.36000
3	0.35467	0.24267	0.40267
$\vdots$	$\vdots$	$\vdots$	$\vdots$
19	0.38364	0.22013	0.39623

This is a good approximation after 20 iterations. Note that for the web this would take thousands of iterations (but is still faster than solving exactly!)

Representation in Memory

We will use structs to help encode this

```
#include <stdio.h>
typedef struct {
  int src, dst;
} link;
int main(void) {
  link l[] = {{0,1}, {0,2}, {1,2}, {2,0}};
  return 0;
}
```



Main Ideas Of Next Slide

- Compute out links
- Make initial guess and store in `r`, normalized
- Perform the iterative process in `s` and update in `r`.

PageRank.c

```
void pagerank (link l[], int n_link, double r[],
  int n_page, double delta, int n_iter){
  double s[n_page];
  int out[n_page];
  for (int i = 0; i < n_page; i++) out[i] = 0;
  for (int j = 0; j < n_link; j++)
    out[l[j].src]++;
  for (int i = 0; i < n_page; i++)
    r[i] = 1.0 / n_page;
  for (int k = 0; k < n_iter; k++) {
      for (int i = 0; i < n_page; i++)
        s[i] = (1.0 - delta) / n_page;
      for (int j = 0; j < n_link; j++)
        s[l[j].dst] +=
          (r[l[j].src]/out[l[j].src])*delta;
      for (int i = 0; i < n_page; i++)
        r[i] = s[i]; }}
```

PageRank.c (Continued)

```
#include <stdio.h>
//Insert page rank code from before here
void main () {
  link l[] = {{0,1},{0,2},{1,0},{2,1}};
  double r[3];
  pagerank(l, sizeof(l)/sizeof(l[0]), r,
    sizeof(r)/sizeof(r[0]), 0.80, 20);
  for (int i = 0; i < 3; i++)
    printf("%g\n", r[i]);
}
```

Final Warning - Sinks

- What happens if a page has no outgoing links?
- We call such pages sinks. With a sink, the only way to escape it is to leave randomly.
- With sink nodes, probabilities become lost.
- Let's see this explicitly with an example.

An Example

Consider the following with $\delta = 0.8$:

$$\text{A} \longrightarrow \text{B}$$

- The page rank equations for this are

$$
\begin{aligned}
r(A) &= \frac{1-\delta}{2} = 0.1 \\
r(B) &= \frac{1-\delta}{2} + \delta \cdot \frac{r(A)}{1} = 0.18
\end{aligned}
$$

- Notice that these values do not add up to 1 and will never change after iterations.

Fixes

We can fix this problem in a few ways:

- Renormalize the final answer (this is bad because it over-estimates the page rank for nodes with many in-links)
- Connect each sink node to all other nodes (this is expensive)
- Connect each sink node without actually connecting them:

$$r_\delta(P) = \frac{1-\delta}{N} + \delta \sum_{i=1}^{M} \frac{r(Q_i)}{|Q_i|} + \delta \sum_{i=1}^{M_0} \frac{r(S_i)}{N}$$

where the first sum contains no sinks and the second sum is over all sinks (M_0) is the total number of sinks.

Other Problems

- What about cyclic links?
- Hoarding - websites linking to each other to boost page rank.
- We won't discuss these (or other issues) in this course.