Programmeertechnieken

Aanvulling op Programmeren

Ben Ruijl

Radboud Universiteit Nijmegen

10 December 2012

Het postkantoorprobleem

Vind de gemiddelde wachttijd in een postkantoor met

- P loketten
- N klanten
- Klanten komen aan tussen 0 en T
- Gem. verwerkingstijd τ, met $P(t) = te^{-t^2}$
- Alle klanten worden afgehandeld

De toestand

De toestand op tijdstip t bestaat uit

- Het aantal mensen in de rij per kassa
- De vertrektijden per kassa

Gebeurtenissen

Er zijn twee type gebeurtenissen

- Een klant komt aan
- Een klant vertrekt

Eindconditie

De simulatie loopt af als er geen gebeurtenissen meer zijn.

Strategieschets

Er moet gebeuren

- Gemiddelde over meerdere simulaties
- Sorteren van gebeurtenissen op tijd
- Alle geordende gebeurtenissen afwerken

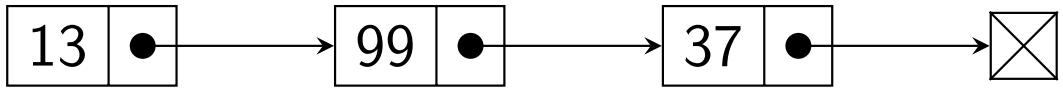
Idee

Maak een altijd gesorteerde queue m.b.v. een linked list

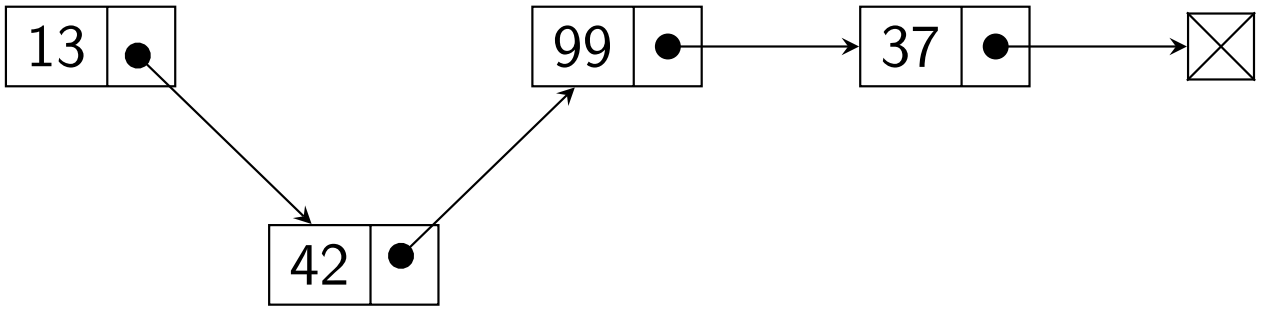
Linked list code

```
1  struct list_elem {
2      int data;
3      struct list_elem* next;
4  };
```

Linked list, grafisch



Element toevoegen



Array vs linked list

Array:

- Gemakkelijk items toevoegen aan het eind
- Snel element i opvragen
- Vaste lengte

Linked list:

- Gemakkelijk items toevoegen in het midden
- Opvragen is lineair
- Variabele lengte

De eventstructuur

```
1 #define NEW_CLIENT 0
2 #define CLIENT_HANDLED 1
3
4 typedef struct list_elem {
5     int type;
6     float time;
7     int counter; /* Loketnummer */
8     struct list_elem* next;
9 } Event;
```

Sorted insert (I)

```
1 Event* sorted_insert(Event* queue, Event* event) {
2     Event* cur = queue;
3
4     /* Moet event vooraan in de queue? */
5     if (cur == NULL || event->time < cur->time) {
6         event->next = cur;
7         return event;
8     }
```

Sorted insert (II)

```
8       while (cur != NULL) {
9           if (cur->next == NULL ||
10           event->time < cur->next->time) {
11              /* Element invoegen */
12              event->next = cur->next;
13              cur->next = event;
14              return queue;
15          }
16
17          cur = cur->next;
18      }
```

De simulatie

```
1   Event* queue = NULL;
2   float* counter_time = malloc(P * sizeof(float));
3   int* counter_queue = malloc(P * sizeof(int));
4
5   float avg = 0;
6   for (j = 0; j < NUM_ITER; j++) {
7       avg += do_simulation();
8   }
9
10  float sol = avg / (float)(N * NUM_ITER);
```

De strategie

Per simulatie:

- Vul de queue met aankomstevents
- Klant bij loket $\rightarrow$ Maak nieuw vertrekevent
- Klant weg bij loket met rij $\rightarrow$ Maak nieuw vertrekevent
- Blijf doorgaan zolang de queue niet leeg is

De aankomsten

Aankomsten kunnen meteen toegevoegd worden

```
1 for (i = 0; i < N; i++) {
2     Event* new_client = malloc(sizeof(Event));
3     new_client->type = NEW_CLIENT;
4     new_client->time = rand() / (float)RAND_MAX * T;
5     new_client->next = NULL;
6
7     queue = sorted_insert(queue, new_client);
8 }
```

De queue loop

```
1   while (queue != NULL) {
2     if (queue->type == NEW_CLIENT) {
3         /* Handel nieuwe client af */
4     }
5     if (queue->type == CLIENT_HANDLED) {
6         /* Handel vertrek af */
7     }
8
9      Event* tmp = queue;
10     queue = queue->next; /* Naar de volgende */
11     free(tmp);
12  }
```

Een nieuwe client

```
1  if (queue->type == NEW_CLIENT) {
2     int min_counter = get_shortest_queue();
3
4     counter_queue[min_counter]++;
5
6     if (counter_queue[min_counter] == 1) {
7        add_handled_event(queue, min_counter);
8     }
9  }
```

De kortste rij

Kortste rij qua aantal mensen:

```
1  int get_shortest_queue() {
2      int shortest = 0;
3      for (i = 1; i < P; i++) {
4          if (counter_queue[i] < counter_queue[shortest]) {
5              shortest = i;
6          }
7      }
8
9      return shortest;
10 }
```

Een afgehandelde client

```
1  if (queue->type == CLIENT_HANDLED) {
2      counter_queue[queue->counter]--;
3      last_time = queue->time;
4
5      if (counter_queue[queue->counter] > 0) {
6          queue = add_handled_event(queue, queue->counter);
7      }
8  }
```

Een handled event toevoegen

```
1 Event* add_handled_event(Event* queue, int counter) {
2     Event* event = malloc(sizeof(Event));
3     event->type = CLIENT_HANDLED;
4     event->time = queue->time + sqrt(-log(rand()
5           / (float)RAND_MAX)) * tau * M_2_SQRTPI;
6     event->counter = counter;
7     event->next = NULL;
8     return sorted_insert(queue, event);
9 }
```

Finis coronat opus

Doel bereikt

Het postkantoorprobleem is opgelost!

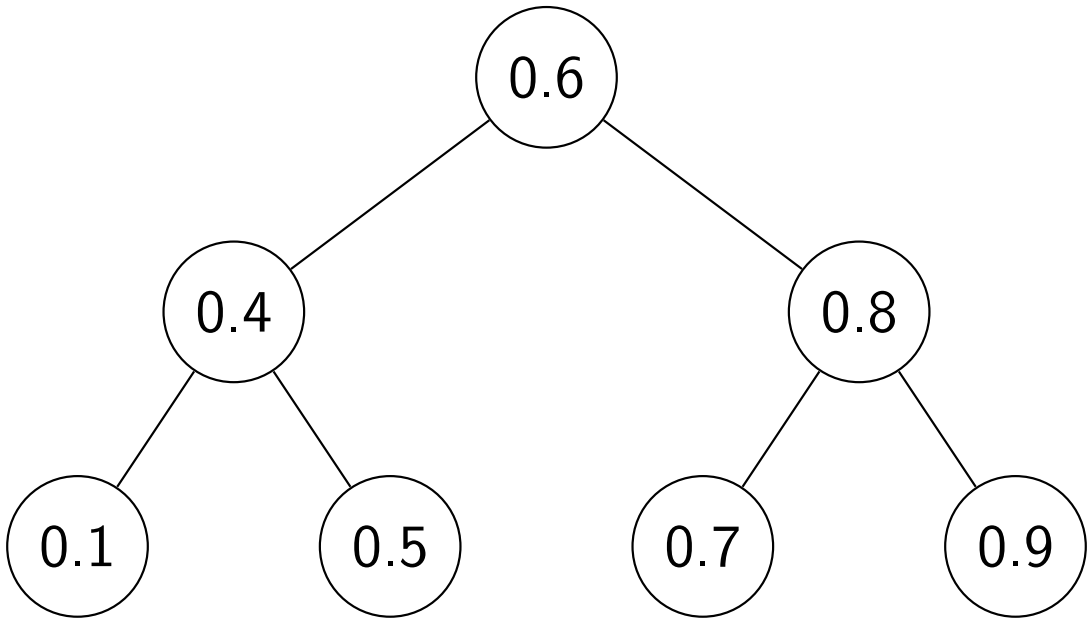
Kleine optimalisaties:

- Events toevoegen gaat lineair met aantal events
- Er is een slimmere manier dan linked lists

Binaire boom

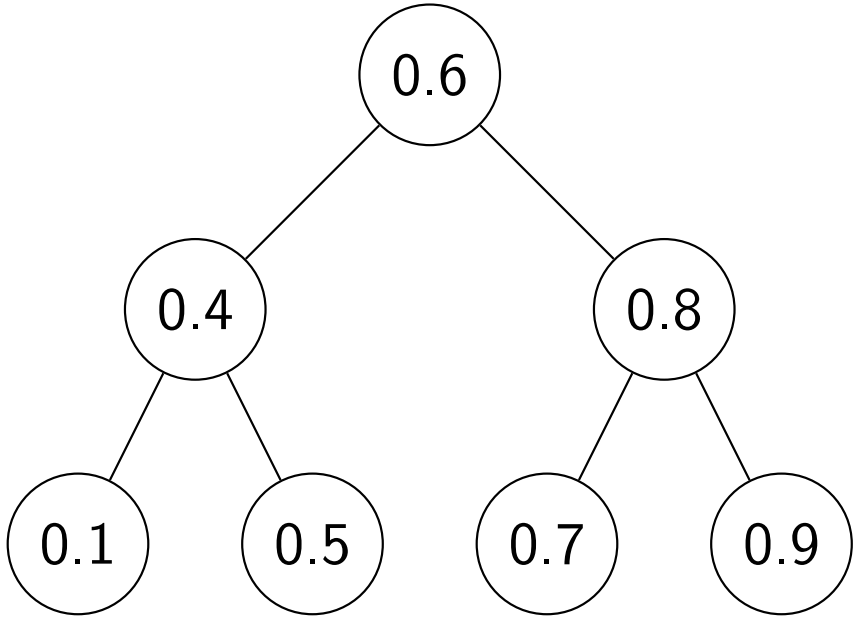
```
1    struct tree_elem {
2         float data;
3         struct tree_elem* left;
4         struct tree_elem* right;
5    };
```

Binaire boom, grafisch



Voor- en nadelen

- Sneller opzoeken
- Sneller toevoegen
- Moeilijk te balanceren



Recursieve functies

Een recursieve functie bestaat uit:

- Een vaste startwaarde / eindwaarde
- Een aanroeping van zichzelf
- Eindwaarde $\rightarrow$ functie eindigt

In de wiskunde:

$$f(0) = 1$$
$$f(n) = n \cdot f(n-1)$$

Voorbeeld in C

De faculteitsfunctie:

```
1    int fac(int n) {
2        if (n == 0)
3            return 1;
4
5        return n * fac(n - 1);
6    }
```

Recursief vs. niet recursief

Recursief:

```
1 int fac(int n) {
2   if (n == 0)
3     return 1;
4
5
6   return n * fac(n - 1);
7 }
```

Niet recursief:

```
1 int fac(int n) {
2     int i, f = 1;
3     for (i = 2; i <= n; i++)
4         f *= i;
5
6     return f;
7 }
```

Fibonacci-serie

$$0\ 1\ 1\ 2\ 3\ 5\ 8\ 13\ 21\ 34\ldots$$

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

Naïeve implementatie

```
1  int fibonacci(int n) {
2      if (n < 2)
3          return n;
4
5      return fibonacci(n - 2) + fibonacci(n - 1);
6  }
```

Problemen en oplossingen

- Fibonacci is erg traag voor grote n
- Er wordt heel veel opnieuw berekend

Idee

f_{n-2} (en dus ook f_{n-1}) kunnen we onthouden!

Betere implementatie

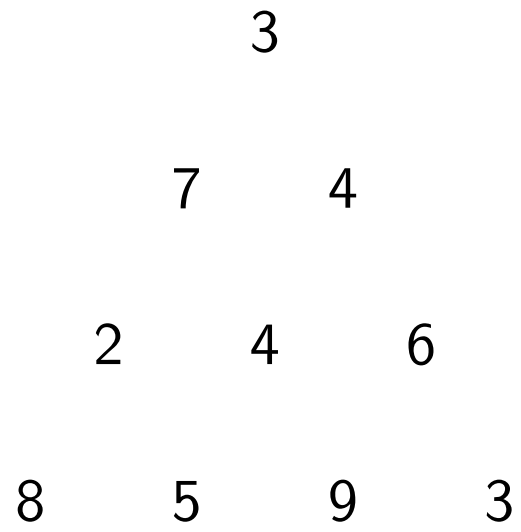
```
1   int fib[200] = {0};
2
3   int fibonacci(int n) {
4       if (n < 2)
5           return n;
6
7       if (fib[n] > 0)
8           return fib[n];
9
10      int fibn = fibonacci(n - 2) + fibonacci(n - 1);
11      fib[n] = fibn;
12      return fibn;
13  }
```

Resultaten

- Antwoord in paar milliseconden!
- Opslaan van subresultaten heet *dynamic programming*
- Snelheid ten koste van geheugen

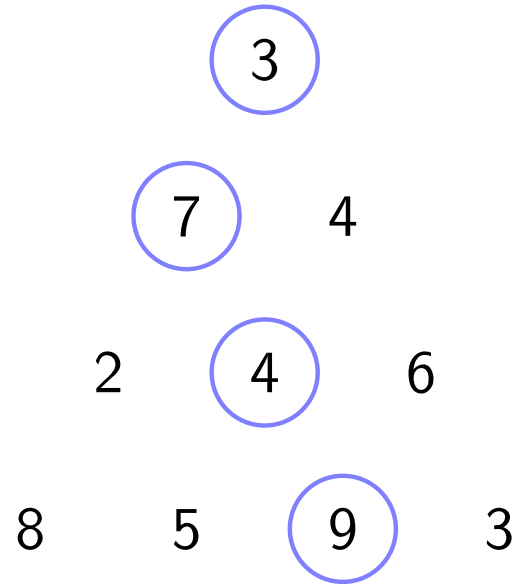
Het driehoekprobleem

- Loop naar beneden
- Zoek pad met hoogste score



Het driehoekprobleem

- Loop naar beneden
- Zoek pad met hoogste score

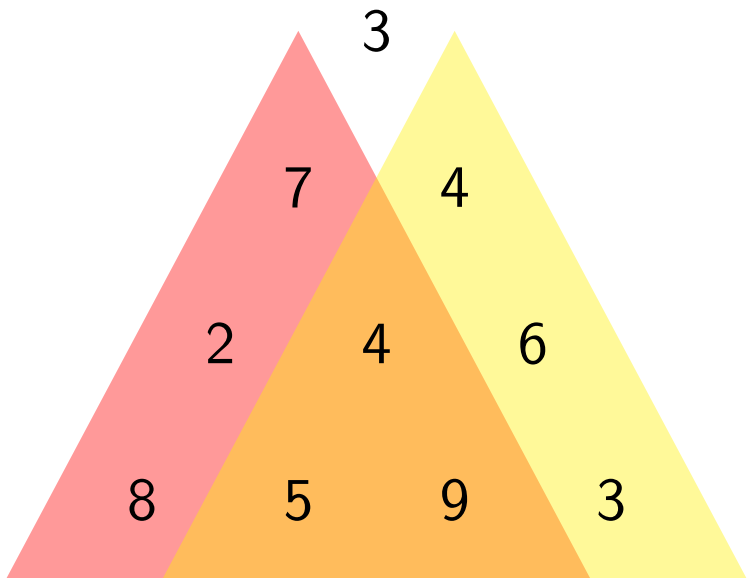


De uitdaging

- Stel de driehoek is 100 lagen diep
- Aantal paden: 2^{99}
- Stel je doet 10^{12} paden per seconde
- Dan duurt het nog steeds meer dan 20 miljard jaar!

Subprobleem

- Waarde bij 3?
- Bereken waarde bij 7
- Bereken waarde bij 4
- Pak maximum



Opslaan van waarden

- Waarde op positie x,y opslaan
- Simpele oplossing kost wat geheugen

```
1    int value[100][100];
```

(0,0)

(0,1) (1,1)

(0,2) (1,2) (2,2)

Oplossing driehoekprobleem

```
1   int get_value(int x, int y) {
2       if (y == 99) /* Laatste rij */
3           return tree[x][y];
4
5       if (value[x][y] > 0)
6           return value[x][y];
7
8       int left = get_value(x, y + 1);
9       int right = get_value(x + 1, y + 1);
10      value[x][y] = tree[x][y] + max(left, right);
11      return value[x][y];
12  }
```

Het substitutiecijfer

Maak een bijectie:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
G	T	U	J	C	P	L	D	O	R	Y	W	X	Z	F	I	V	Q	S	K	A	N	E	M	H	B

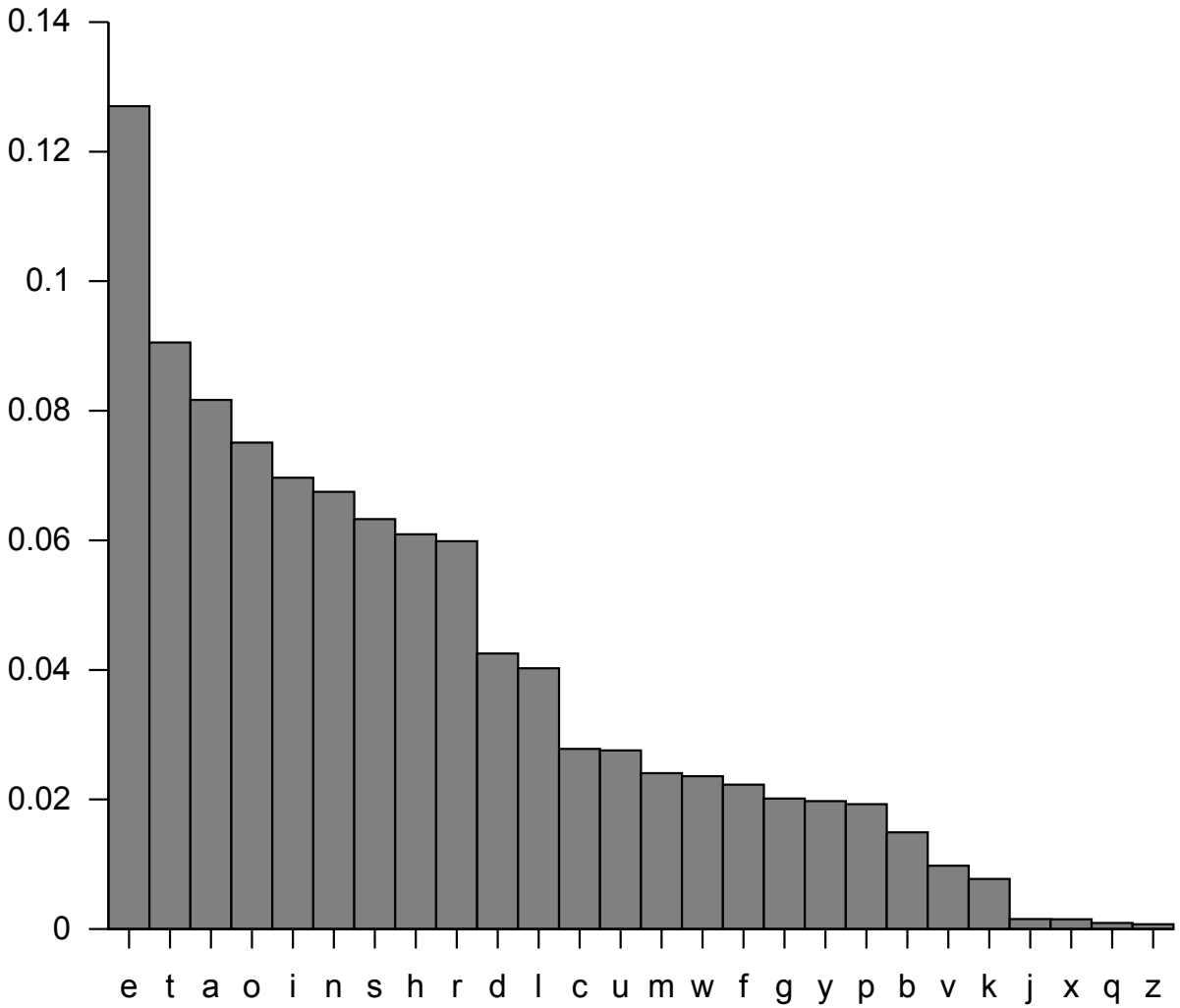
Vertaal:

- HALLO $\rightarrow$ DGWWF
- WERELD $\rightarrow$ ECQCWJ

De uitdaging

- Aantal mogelijkheden: $26! \approx 4 \cdot 10^{26}$
- Hoe vinden we de originele tekst?
- Welke eigenschap van de brontekst blijft bewaard?

Letterfrequenties



Tetragrammen

- Letterfrequenties zijn een goed begin
- Tetragramfrequenties zijn stabieler
- DITISEENZIN splitst op in
  - DITI, ITIS, TISE, ISEE, SEEN, EENZ, ENZI, NZIN
- Bereken frequenties uit grote boeken (bijbel)

Strategie

Een tekst lijkt meer op Nederlands als de tetragramfrequenties overeenkomen!

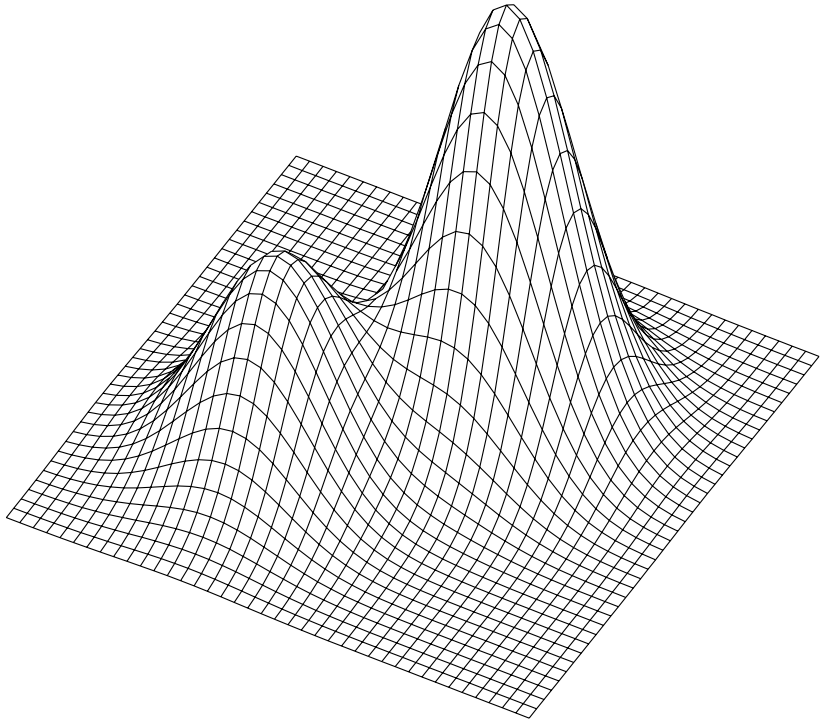
- We kunnen dus een scorefunctie maken
- Tel de frequenties f in tekst
- Vergelijk met de referentie r

Score

$$S = \sum_i e^{-\frac{(f_i - r_i)^2}{2\sigma^2}}$$

Oplossingsruimte

- We zoeken oplossing met hoogste score
- Oplosruimte is te groot om alles af te gaan



Hill climbing

- We zoeken absoluut maximum
- Stel we beginnen op een random plaats
- Zoek een pad naar boven

Idee

Bekijk buren van de huidige locatie en loop naar buur die beter is

Hill climbing code

```
1  void hillclimb(char* text, char* alpha) {
2      int i, j;
3      float best_score = 0;
4
5      for (i = 0; i < 25; i++)
6          for (j = i + 1; j < 26; j++) {
7              swap(alpha, i, j);
8              float new_score = score(transcribe(text, alpha));
9              if (new_score > best_score) {
10                 i = j = 0;
11                 best_score = new_score;
12             } else
13                 swap(alpha, i, j);
14         }
15 }
```

Einde

"Computer science education cannot make anybody an expert programmer any more than studying brushes and pigment can make somebody an expert painter."

(Eric Raymond)