Ausgewählte Probleme aus dem ACM Programming Contest

Dozentin: Doina Logofătu

Von: Barny Porcio

Inhaltsverzeichnis

1. Problem: Burger (557)

Methoden:

- Wahrscheinlichkeitsrechnung
- Dynamische Programmierung

Problemset

- Geburtstagsfeier von Bill und Ben bei McDonald's
- Gesamt (inklusive Bill und Ben) **20** Personen
- Ronald McDonald hat je **10** Cheeseburger **C** und 10 Hamburger **H** zubereitet

Problemset

- Ronald McDonald verteilt die Burger reihum, Bill und Ben werden als letztes bedient
- Damit die Bürger gerecht verteilt werden wird bei jeder Person eine (faire)Münze geworfen, Kopf heißt Hamburger, Zahl heißt Cheeseburger
- Als Ronald bei Ben und Bill angelangt ist, muss er die Münze nicht mehr werfen, da nur noch 2 Hamburger übrig geblieben sind.

Problemset

- Ronald wundert sich sehr über diesen Ausgang, und möchte gern wissen, wie wahrscheinlich es ist, dass solch ein Ergebnis bei **N** Personen auftritt.
- (N darf gerade Werte bis zu 100.000 annehmen)

Lösungsweg

- etwas schwierig direkt auszurechnen
- → Besser: über die Gegenwahrscheinlichkeit d. h. gesucht ist die Wahrscheinlichkeit dafür, dass Bill und Ben unterschiedliche Bürger bekommen.

Beispiel für N = 6

- Gesucht ist die Wahrscheinlichkeit für

  _ _ _ _ C H
- oder
- _ _ _ _ H C

C = Cheeseburger
H = Hamburger
_ = C oder H

- wobei wir einfach Voraussetzen das die letzten 2 Burger verschieden sind. Da bis zum vorletzten Bürger Noch beide Bürger vorhanden sind ist bis dahin die Wahrscheinlichkeit pro Bürger 0.5, beim letzten Bürger aber gibt es keine Andere Wahl mehr, somit ist dort die Wahrscheinlichkeit 1

Beispiel für N = 6

- Die Wahrscheinlichkeit für eine mögliche Konstruktion ist also:
- 0.5 * 0.5 * 0.5 * 0.5 * 0.5 * 1
- für z. B. (Variabel) (Fix) C C H H C H

Den Fall H C können wir uns an dieser Stelle sparen indem wir die Wahrscheinlichkeit für C H mit 2 multiplizieren (H C hat dieselbe Wahrscheinlichkeit wie C H). Wenn wir das gleich in unserer Formel berücksichtigen bekommen wir 0.5 * 0.5 * 0.5 * 0.5 * 1 * 1 * **6**

Nur eine Möglichkeit, auch möglich ist:

C H C H
C H H C
H C H C
H C C H
H H C C

Macht insgesamt 6 Möglichkeiten

Gegenwahrscheinlichkeit = 0.375 →
Wahrscheinlichkeit = 1 – 0.375 = **0.625**

Für N

- Um die Aufgabe für N zu Lösen braucht man die Wahrscheinlichkeit für einen möglichen Fall: $0.5^{(N-2)}$
- Und die Anzahl der möglichen Verteilungen der Bürger(ohne die 2 letzten). Hierfür eignet sich der Binominialkoeffizient $\binom{N-2}{(N-2)/2}$

Wiederholung

$$\binom{n}{k} = \frac{n!}{k! * (n-k)!}$$

Resultat: $1 - \left[(0.5)^{(N-2)} * \binom{N-2}{(N-2)/2} \right]$

Problem gelöst?

- Grundsätzlich wäre hiermit das Problem schon gelöst, einziges Problem: Maximal möglicher Binominialkoeffizient = $\binom{99998}{49999}$
- Selbst mit BigInteger nicht möglich da der Java heap space zu klein ist

Lösung des Speicherproblems

- Man arbeitet mit dem Logarithmus
- $\text{Log}\left(\frac{N!}{K! * (N-K)!}\right) = \text{Log}(N!) - \text{Log}(K!) - \text{Log}((N-K)!)$
- Dabei empfiehlt es sich, wie wir später sehen werden, den Log zur Basis 2 zu wählen.
- Um nun den Binominialkoeffizienten wieder raus zu bekommen müssen wir 2 mit dem Log potenzieren: $\binom{N}{K} = 2^{\text{Log}\binom{N}{K}}$

Lösung

- Nun lautet unsere Gesamte Lösungsformel

$$0.5^{(N-2)} * 2^{\text{Log}\left(\left(\frac{N-2}{(N-2)/2}\right)\right)}$$

$$\Rightarrow 0.5^{(N-2) - \text{Log}\left(\frac{N-2}{(N-2)/2}\right)}$$

Programmcode

```
//wird öfter benötigt
final static double log_of_2 = Math.log(2);

//enthält den Log2 vom Index Fakultät
static double[] log2_fak = new double[99999];
//array auffüllen
for (int i = 1;i<log2_fak.length;++i){
    log2_fak[i]= log2_fak[i-1]+Math.log(i)/log_of_2;

}


static public double berechnung(int n){
    double log2_of_n_over_k =  log2_fak[n-2]
                              -log2_fak[(n-2)/2]
                              -log2_fak[(n-2)/2];
    return 1-(Math.pow(0.5, (n-2)-log2_of_n_over_k));
}
```

Alternative Lösung

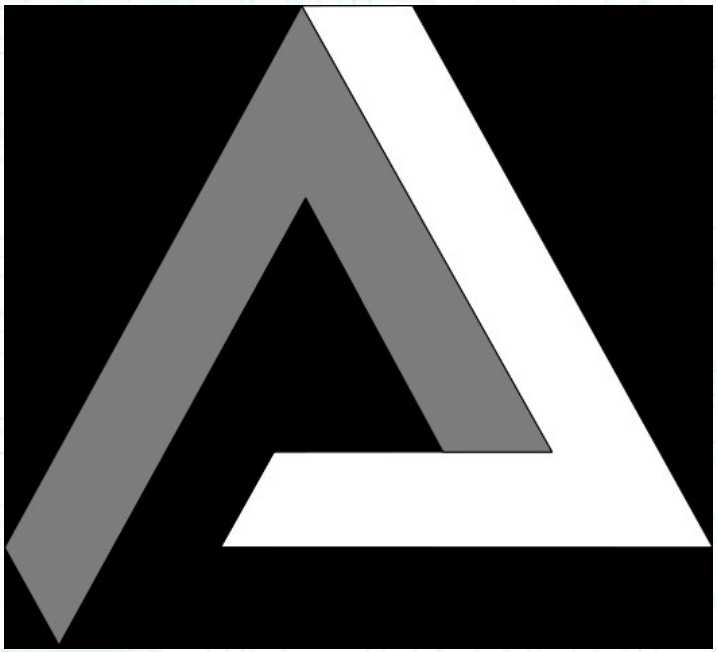
```
#include <iostream>
#include <vector>

 const int NMAX = 50000;
 using namespace std;

 int main(){
      long int n, i, k;
      vector<long double> T;
      T.push_back(1);
      for(i=1; i<=NMAX; i++){
            T.push_back((long double)((2*i-1.0)/(2*i)*T[i-1]));
      }
      cin>>n;
      for(i=0; i<n; i++){
            cin >> k;
            cout.precision(4);
            cout.flags(ios::fixed);
            cout  <<  1-T[k/2-1] << endl;
      }
      return 0;
}
```

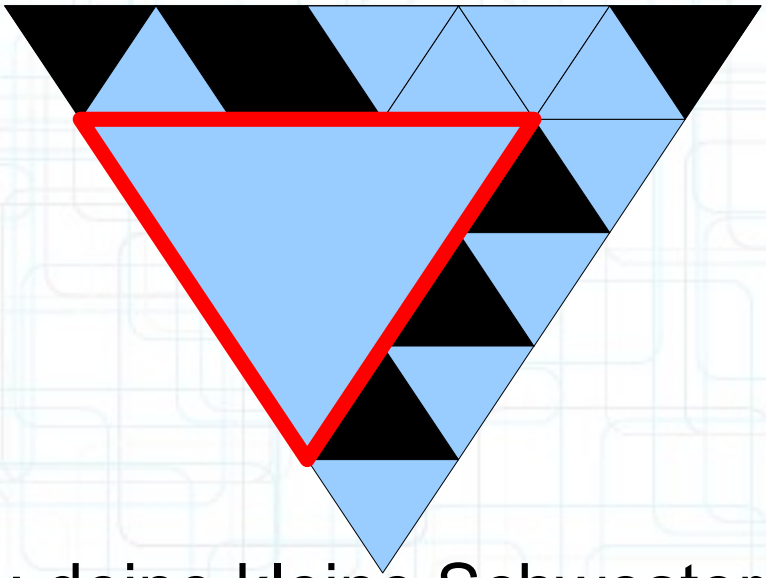
2. Problem: Triangles (585)

- Methoden:
  - Brute-Force (iterativ)



Problemset

- Du hast die Ehre dieses Jahr den Weihnachtsstern für euren Christbaum zu basteln. Dafür brauchst du ein großes Stück dreieckiges Papier



- Problem: deine kleine Schwester hat für kleinere Sterne schon eine Menge kleinerer Dreiecke ausgeschnitten
- Gesucht: Das größte Dreieck das man aus dem restlichen Papier ausschneiden kann

Problemset

- Gegeben ist N, die Höhe des großen Dreiecks
- Darauf folgen N Strings die die jeweilige Stelle

```
5
#_##____#
 _____#_
  ___#_
   _#_
    _
```

 # = Loch
_ = Papier
space = platzhalter

- Gesucht: Das größte Dreieck ohne Loch. In diesem Dreieck: 9

Lösungsweg

- Zuerst lesen wir den Input und schreiben das Dreieck in ein zuvor erstelltes 2 dimensionales boolean Array, in dem true stehen soll wenn an der stelle Papier vorhanden ist

```
//2D Array, 1.[]stellt die Zeile des Dreiecks dar, das 2.[] das Zeichen(Spalte)

//1.[n-1] ist die oberste reihe 2.[0] das linkeste element.

static boolean[][] triangle;

//Code zum Anfüllen
```

Lösungsweg

- Als nächstes entwirft man eine Methode, die für eine gegebene Zeile und Spalte das von dort aus größte Dreieck berechnet und deren Größe zurück liefert. Es ist zu beachten, das jedes kleine Dreieck nur in eine Richtung (entweder nach oben o. nach unten) überprüft werden kann. In welche Richtung gelaufen werden soll wird mit einem boolean übergeben.

```
static int calculate(boolean up, int stufe, int mittelposition){
    if (up){
        //geht je eine stufe bis zur maximalen stufe hoch
        for(int i= stufe; i<triangle.length;++i){
            //geht die spalten einer reihe entlang und schaut ob es eine luecke
dazwischen gibt
            for(int position = mittelposition-(i-stufe);
                    position<=mittelposition+(i-stufe);++position){
                if (!triangle[i][position])
    //Wenn es eine Lücke gibt wird das bis dahin größte Dreieck zurückgeliefert
                    return (i-stufe) * (i-stufe);
            }
        }
        //wenn es bis nach oben durchging muss das groesste dreieck noch gesetzt
werden, da es nicht aufgehalten wurde muss es bei der maximalen laenge triangle-lengt
aufgehoert haben
        return (triangle.length-stufe) * (triangle.length-stufe);
    }
```

fast genau das selbe brauchen wir auch noch für den Fall das es nach unten überprüft werden muss!

Lösungsweg

- Nun müssen wir nur noch durch unser großes Dreieck wandern, jeweils für jedes kleine Dreieck die Methode mit dem Richtigen boolean aufrufen und überprüfen, ob der neu errechnete Wert großer ist als unser bis dahin gefundene größter Wert
- Das erste kleine Dreieck in jeder Zeile kann immer nur noch oben Überprüft werden, die darauf folgenden sind immer abwechselnd nach unten und dann wieder nach oben laufende Dreiecke.

```
//geht das komplette dreieck durch und ueberprueft von bei jedem '-' mit der
methode calculate
        //das groest moegliche dreieck an der stelle und aendert
biggesttriangle wenn noetig ab
int biggesttriangle = 0;
        for(int i = triangle.length-1;i >= 0;--i){
            boolean status = false;
            for(int i2 = (triangle.length-1)-i;
            i2 <=(triangle[0].length-1)-(triangle.length-1-i) ; ++i2){
                status = !status;
                if (triangle[i][i2]){
                    int newcalc = calculate(status, i, i2);
                    if (biggesttriangle < newcalc)
                        biggesttriangle = newcalc;
                }
            }
        }
```

An dieser Stelle muss nur noch biggesttriangle ausgegeben werden

Programmcode

Siehe Eclipse

Vielen Dank
für eure
Aufmerksamkeit!