РАНДОМИЗИРАНИ АЛГОРИТМИ

Рандомизираните алгоритми използват случайни числа. Генераторите на случайни числа са детерминирани алгоритми, т.е. числата са псевдослучайни, което обикновено не е пречка.

Рандомизираните алгоритми се делят на два класа:

— Алгоритми от тип Лас Вегас: винаги дават правилен резултат, но времето за работа е случайна величина. Тоест, ако алгоритъм от тип Лас Вегас бъде пуснат два пъти върху едни и същи входни данни, резултатът и двата пъти ще бъде един и същ, но времето за изпълнение на алгоритъма може да е различно.

П р и м е р 1: Ако сортиращ алгоритъм на всяка стъпка избира за сравняване елементи със случайни индекси, то изходът винаги ще е правилно сортиран масив, но времето за работа ще е различно при всяко изпълнение на алгоритъма.

В общия случай времето на алгоритъм от тип Лас Вегас е неограничено. Допуска се такъв алгоритъм да работи безкрайно, но обикновено се изисква математическото очакване на времето да бъде крайно.

Със **ZPP** се означава класът на всички задачи за разпознаване (decision problems), за които съществува алгоритъм от тип Лас Вегас с полиномиална времева сложност в най-лошия случай. Тъй като времето за изпълнение (при дадена дължина на входа) зависи както от конкретните входни данни, така и от случайните избори, които прави алгоритъмът, то трябва да уточним, че думите "най-лошия случай" се отнасят за входните данни (т.е. по тях се взима максимумът на времето), а по случайните избори на алгоритъма се взима средна стойност (т. нар. математическо очакване).

— Алгоритми от тип Монте Карло: времето им е ограничено отгоре от детерминирана функция, но резултатът е случаен, т.е. може да е грешен с известна вероятност.

П р и м е р 2: Рандомизиран алгоритъм за пресмятане на лицето на сечението на n кръга с центрове $(x_k\ ;\ y_k)$ и радиуси R_k, $k = 1,\ 2,\ 3,\ \ldots,\ n$.

```
CirclesIntersection(X[1...n], Y[1...n], R[1...n])
minX ← min(X[k]-R[k]);  maxX ← max(X[k]+R[k])   (k = 1...n)
minY ← min(Y[k]-R[k]);  maxY ← max(Y[k]+R[k])   (k = 1...n)
count ← 0
for k ← 1 to 500
   X0 ← rand([minX ; maxX]);   Y0 ← rand([minY ; maxY])
   if CheckPoint(X0 , Y0 , X[1...n], Y[1...n], R[1...n])
      count ← count + 1
return (count / 500) × (maxX − minX) × (maxY − minY)

CheckPoint(X0 , Y0 , X[1...n], Y[1...n], R[1...n])
for k ← 1 to n
   if (X0 − X[k])^2 + (Y0 − Y[k])^2 > (R[k])^2
      return false
return true
```

Алгоритъмът се изпълнява за най-много $504\,n$ стъпки, т.е. времето му е $\Theta(n)$. Кръговете се ограждат с правоъгълник П и за лице на сечението се взима частта от лицето на П, пропорционална на частта на случайните точки, попаднали в сечението.

BPP е класът на задачите за разпознаване, за които има рандомизиран алгоритъм, който дава верен отговор с вероятност поне $2/3$ (следователно вероятността за грешен отговор не надхвърля $1/3$) и времето за работа на алгоритъма (което може да е случайна величина) е ограничено отгоре от полином на n — дължината на входа. Този полином не зависи нито от случайните избори, които алгоритъмът прави, нито от конкретния вход (а зависи само от дължината n на входа).

Анализът на рандомизираните алгоритми се основава на теорията на вероятностите.

Някои сведения от теорията на вероятностите

Вероятност на събитие се нарича частта на изходите, благоприятни за това събитие, спрямо броя на всички изходи:

$$\mathbf{P}(A) = \frac{k_A}{k_\Omega} \quad \text{(класическа вероятност).}$$

Тук $\mathbf{P}(A)$ е вероятността на случайното събитие A;

k_A е броят на изходите, благоприятни за събитието A;

k_Ω е броят на всички изходи — благоприятни и неблагоприятни.

Благоприятни са тези изходи, при които събитието се сбъдва (настъпва).

Понеже $0 \le k_A \le k_\Omega$, то $0 \le \mathbf{P}(A) \le 1$. В проценти: $0\% \le \mathbf{P}(A) \le 100\%$.

Събитията се разглеждат като множества от елементарни изходи.

Невъзможно събитие: $\varnothing$ (не съдържа изходи).

Сигурно събитие: Ω (съдържа всички изходи).

Върху събития може да се извършват операции от теорията на множествата:

— допълнение: съответства на отрицание (“не”), т.е. допълваното събитие не настъпва;

— сечение: съответства на конюнкция (“и”), т.е. настъпват всички събития;

— обединение: съответства на дизюнкция (“или”), т.е. настъпва поне едно събитие.

Условна вероятност $\mathbf{P}(A \mid B)$ е вероятността да настъпи събитието A, при условие че е настъпило събитието B. Изисква се предположението да е осъществимо, т.е. $B \neq \varnothing$.

Свойства на вероятността:

1) $\mathbf{P}(\varnothing) = 0$. 2) $\mathbf{P}(\Omega) = 1 = 100\,\%$. 3) $\mathbf{P}\left(\overline{A}\right) = 1 - \mathbf{P}(A)$.

4) $\mathbf{P}(A_1 \cap A_2 \cap \ldots \cap A_n) = \mathbf{P}(A_1)\ \mathbf{P}(A_2)\ \ldots\ \mathbf{P}(A_n)$, ако събитията са независими в съвкупност (т.е. сбъдването на кое да е от тях не влияе върху останалите събития).
$\mathbf{P}(A_1 \cap A_2 \cap \ldots \cap A_n) = \mathbf{P}(A_1)\ \mathbf{P}(A_2 \mid A_1)\ \mathbf{P}(A_3 \mid A_1 \cap A_2)\ \ldots\ \mathbf{P}(A_n \mid A_1 \cap \ldots \cap A_{n-1})$
при зависими събития (т.е. за всяко събитие се взимат предвид предишните събития).

5) $\mathbf{P}(A_1 \cup A_2 \cup \ldots \cup A_n) = \mathbf{P}(A_1) + \mathbf{P}(A_2) + \ldots + \mathbf{P}(A_n)$, ако събитията са две по две несъвместими (т.е. съвместното сбъдване на кои да е две от тях е невъзможно). Ако пък има съвместими събития, прилага се принципът за включване и изключване:

$$\mathbf{P}(A_1 \cup A_2 \cup \ldots \cup A_n) = \sum_{\varnothing \neq X \subseteq \{1,\,2,\,\ldots,\,n\}} \left((-1)^{|X|+1}\ \mathbf{P}\left(\bigcap_{k \in X} A_k \right) \right).$$

Дискретни случайни величини

Всяка случайна величина има закон за разпределение, който задава съвкупността от възможни стойности на величината и вероятността за всяка стойност. Тук разглеждаме само дискретните случайни величини, т.е. тези величини, чието множество от стойности е изброимо (крайно или безкрайно).

Нека случайната величина ξ има следното дискретно разпределение:

стойност	x_1	x_2	...	x_n
вероятност	p_1	p_2	...	p_n

Стойностите x_1 , x_2 , . . . , x_n трябва да бъдат две по две различни.

Вероятностите p_1 , p_2 , . . . , p_n трябва да бъдат неотрицателни числа, а сборът им трябва да бъде равен на единица: $p_1 + p_2 + \ldots + p_n = 1$.

Математическо очакване: $\mathbf{E}\xi = p_1 x_1 + p_2 x_2 + \ldots + p_n x_n$. То има смисъл на средна стойност на случайната величина: ако извършим дълга редица от наблюдения върху стойностите на ξ и пресметнем средноаритметичната стойност на всички наблюдавани, то тя с голяма вероятност ще бъде близка до $\mathbf{E}\xi$.

Дисперсия: $\mathbf{D}\xi = p_1 (x_1 - \mathbf{E}\xi)^2 + p_2 (x_2 - \mathbf{E}\xi)^2 + \ldots + p_n (x_n - \mathbf{E}\xi)^2$. Дисперсията е мярка за средното отклонение на случайната величина от нейното математическо очакване. От определението се вижда, че дисперсията винаги е неотрицателна. Мерната единица на дисперсията е квадратът на мерната единица на самата случайна величина. Затова, въпреки че дисперсията е мярка за отклонението, тя не съвпада със самото отклонение: то се получава, като коренуваме дисперсията:

$\sigma = \sqrt{\mathbf{D}\xi}$ (стандартно отклонение). То винаги е неотрицателно. По определение то е средно (по-точно, средноквадратично) отклонение, а не максимално отклонение.

Свойства на математическото очакване, дисперсията и стандартното отклонение:

1) $\mathbf{E}c = c$. 2) $\mathbf{E}(c\xi) = c \,.\, \mathbf{E}\xi$. Тук c е произволна константа.
В частност, при $c = -1$ получаваме равенството $\mathbf{E}(-\xi) = -\mathbf{E}\xi$.
3) $\mathbf{E}(\xi_1 \xi_2 \ldots \xi_n) = (\mathbf{E}\xi_1)(\mathbf{E}\xi_2) \ldots (\mathbf{E}\xi_n)$. Тази формула важи само за случайни величини, които са независими в съвкупност.
4) $\mathbf{E}(\xi_1 + \xi_2 + \ldots + \xi_n) = \mathbf{E}\xi_1 + \mathbf{E}\xi_2 + \ldots + \mathbf{E}\xi_n$ за всякакви случайни величини.
В частност, $\mathbf{E}(\xi_1 + \xi_2) = \mathbf{E}\xi_1 + \mathbf{E}\xi_2$ и $\mathbf{E}(\xi_1 - \xi_2) = \mathbf{E}\xi_1 - \mathbf{E}\xi_2$.
5) $\mathbf{D}c = 0$. 6) $\mathbf{D}(c\xi) = c^2 \,.\, \mathbf{D}\xi$. Тук c е произволна константа.
В частност, при $c = -1$ получаваме равенството $\mathbf{D}(-\xi) = \mathbf{D}\xi$.
7) $\mathbf{D}(\xi_1 + \xi_2 + \ldots + \xi_n) = \mathbf{D}\xi_1 + \mathbf{D}\xi_2 + \ldots + \mathbf{D}\xi_n$. Тази формула важи само за случайни величини, които са независими в съвкупност.
В частност, $\mathbf{D}(\xi_1 \pm \xi_2) = \mathbf{D}\xi_1 + \mathbf{D}\xi_2$, ако ξ_1 и ξ_2 са независими величини.
8) $\mathbf{D}\xi = \mathbf{E}(\xi^2) - (\mathbf{E}\xi)^2$. Така по-удобно се пресмята дисперсията.
9) $\sigma(c) = 0$. 10) $\sigma(c\xi) = |c| \,.\, \sigma(\xi)$. Тук c е произволна константа.
11) $\mathbf{E}(\xi + c) = c + \mathbf{E}\xi$. 12) $\mathbf{D}(\xi + c) = \mathbf{D}\xi$. 13) $\sigma(\xi + c) = \sigma(\xi)$.

Често срещани дискретни разпределения

Следните дискретни разпределения имат голямо теоретическо и практическо значение:

1) Биномно разпределение: $\xi \sim \text{Bi}(n,\ p) \ \Leftrightarrow\ \xi$ = броят на успехите от n независими опита, всеки от които има една и съща вероятност за успех p.

2) Геометрично разпределение, започващо от нула: $\xi \sim \text{Ge}(p) \ \Leftrightarrow\ \xi$ = броят на неуспехите до първия успех, ако опитите са независими и имат една и съща вероятност за успех p.

3) Геометрично разпределение, започващо от единица: ξ = брой опити до първия успех включително, ако опитите са независими и имат една и съща вероятност за успех p.

   Когато опитите са независими и вероятността за успех е една и съща, се казва, че опитите се провеждат по урновата схема "със връщане" (например трите разпределения по-горе).

4) Хипергеометрично разпределение: $\xi \sim \text{HG}(n, N, M) \ \Leftrightarrow\ \xi$ = брой успехи от n опита по урновата схема "без връщане"; отначало урната има N елемента и M от тях са успехи.

5) Поасоново разпределение: $\xi \sim \text{Po}(\lambda) \ \Leftrightarrow\ \xi$ = брой събития в даден интервал от време, настъпващи поединично, случайно и независимо; λ е средният брой събития, настъпващи в интервал от време със същата дължина.

6) Равномерно разпределение U (n): стойностите са 1, 2, 3, **. . .** , n и са равновероятни.

Разпределение	Закон на разпределение $\mathbf{P}\{\xi = k\}$	Математическо очакване $\mathbf{E}\xi$	Дисперсия $\mathbf{D}\xi$
Биномно $\xi \sim \text{Bi}(n,\ p)$ $n \in \mathbb{N},\ \ 0 < p < 1$	$\binom{n}{k} p^k (1-p)^{n-k}$ $k = 0,\ 1,\ 2,\ \ldots\ ,\ n$	np	$np(1-p)$
Геометрично от 0 $\xi \sim \text{Ge}(p)$ $0 < p < 1$	$(1-p)^k p$ $k = 0,\ 1,\ 2\ \ldots$	$\dfrac{1-p}{p}$	$\dfrac{1-p}{p^2}$
Геометрично от 1 $0 < p < 1$	$(1-p)^{k-1} p$ $k = 1,\ 2,\ 3\ \ldots$	$\dfrac{1}{p}$	$\dfrac{1-p}{p^2}$
Хипергеометрично $\xi \sim \text{HG}(n, N, M)$ $n, N, M \in \mathbb{N};$ $n < N,\ \ M < N$	$\dfrac{C_M^k \cdot C_{N-M}^{n-k}}{C_N^n}$ $\max(0\,;\, n+M-N) \le k \le \min(n\,;\, M)$	$\dfrac{nM}{N}$	$\dfrac{nM(N-M)(N-n)}{N^2(N-1)}$
Поасоново $\xi \sim \text{Po}(\lambda)$ $\lambda > 0$	$\dfrac{\lambda^k}{k!} e^{-\lambda}$ $k = 0,\ 1,\ 2\ \ldots$	λ	λ
Равномерно $\xi \sim \text{U}(n)$ $n \in \mathbb{N}$	$\dfrac{1}{n}$ $k = 1,\ 2,\ \ldots\ ,\ n$	$\dfrac{n+1}{2}$	$\dfrac{n^2-1}{12}$

Приложения на дискретните разпределения при рандомизираните алгоритми

Пример 3: Обикновено генераторът на случайни числа `rand()` поражда дробно число с равномерно разпределение в $[0\,;1)$, само че това равномерно разпределение е непрекъснато, а не дискретно. Понякога обаче имаме нужда от случайно число в някой друг интервал. Това се постига с помощта на линейно преобразуване.

Ако A и B са реални числа (може и дробни), то изразът `A × rand() + B` връща случайна стойност с непрекъснато равномерно разпределение между B (вкл.) и $A + B$.

Ако A и B са цели числа, то изразът $\lfloor$`A × rand() + B`$\rfloor$ връща случайна стойност с дискретно равномерно разпределение между B (вкл.) и $A + B$.

Ако $A = n$ е цяло положително число, а $B = 1$, то $\lfloor$`n × rand() + 1`$\rfloor$ връща случайна стойност с дискретно равномерно разпределение от 1 (вкл.) до n (вкл.), тоест $\mathrm{U}(n)$.

Пример 4: Поасоново разпределение имат величини като следните:
— брой съобщения, изпратени към уебсървър за една секунда;
— брой новооткрити адреси за електронна поща в рамките на един ден;
— брой документи, въведени в счетоводна програма за един месец.
Тази информация е важна при планиране на софтуера.

Задача 1. Даден уебсървър приема средно по 15000 заявки в секунда. Пресметнете вероятността да постъпят поне три заявки през следващата милисекунда.

Решение: Нека ξ = броят на заявките през следващата милисекунда. Тогава $\xi \sim \mathrm{Po}(\lambda)$ и $\lambda = 15000 : 1000 = 15$ заявки в милисекунда (средно). По формулата

$$\mathbf{P}(\xi \geq 3) = \mathbf{P}(\xi = 3) + \mathbf{P}(\xi = 4) + \mathbf{P}(\xi = 5) + \ldots$$

не можем лесно да намерим търсената вероятност, защото вдясно стои безкрайна сума. Затова изваждаме от 100 % останалите случаи:

$$\mathbf{P}(\xi \geq 3) = 1 - \big(\mathbf{P}(\xi = 0) + \mathbf{P}(\xi = 1) + \mathbf{P}(\xi = 2)\big) =$$

$$= 1 - e^{-15}\left(\frac{15^0}{0!} + \frac{15^1}{1!} + \frac{15^2}{2!}\right) \approx 0{,}999961 = 99{,}9961\,\%.$$

Отговор: Има приблизително 99,9961 % вероятност да постъпят три или повече заявки през следващата милисекунда.

Пример 5:
Цикъл по брояч.

```
cnt ← 0
for k ← 1 to n
   if (условие)
      cnt ← cnt + 1
print cnt
```

Отпечатаната стойност има разпределение $\mathrm{Bi}(n, p)$;
n = броя на итерациите,
p = вероятността условието да върне истина.

Пример 6:
Цикъл с предусловие.

```
cnt ← 0
while (условие) do
   обработка на данни
   cnt ← cnt + 1
print cnt
```

Отпечатаната стойност (броят на итерациите) има разпределение $\mathrm{Ge}(p)$ от 0;
p = вероятността условието да върне неистина.

Пример 7:
Цикъл със следусловие.

```
cnt ← 0
repeat
   обработка на данни
   cnt ← cnt + 1
until (условие)
print cnt
```

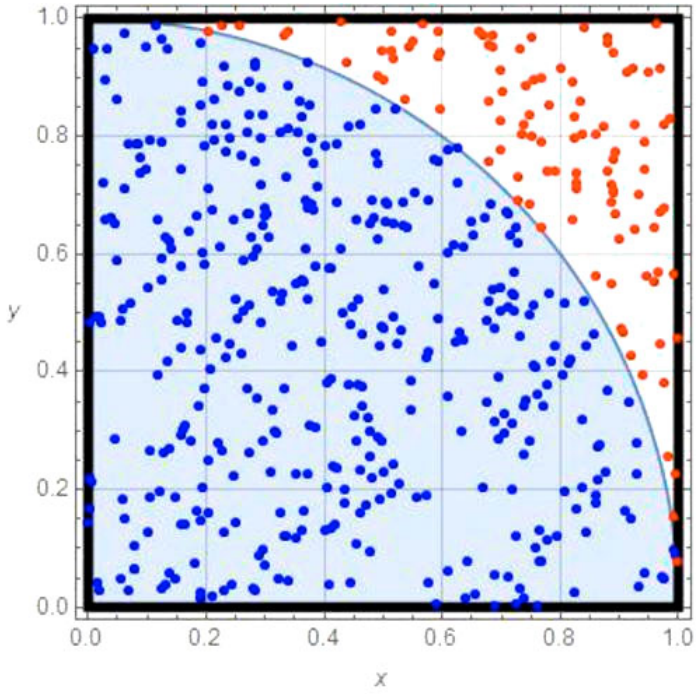
Броят на итерациите има разпределение $\mathrm{Ge}(p)$ от 1;
p = вероятността условието да върне истина.

Задача 2. Каква стойност отпечатва следният програмен код?

```
count ← 0
for k ← 1 to 30000
   if rand()^2 + rand()^2 < 1
      count ← count + 1
print count / 7500
```

Решение: Нека ξ = стойността на променливата `count` след излизане от цикъла. Предполагаме, че функцията `rand()` генерира независими случайни числа, равномерно разпределени в интервала $[0\,;\,1)$. Нека x и y са случайните числа на третия ред от кода. Тогава наредената двойка $(x\,;\,y)$ е равномерно разпределена в единичния квадрат $[0\,;\,1)^2$.

Всяка проверка на неравенството $x^2 + y^2 < 1$ е един опит. Имаме цикъл по брояч, затова броят n на опитите е известен предварително: $n = 30000$. Един опит е успешен $\Leftrightarrow$ $x^2 + y^2 < 1$, т.е. когато точката $(x\,;\,y)$ принадлежи на четвъртината от единичния кръг, намираща се в първи квадрант. Вдясно са показани множество опити (точки): сините точки съответстват на успешни опити, а червените — на неуспешни. Понеже кръгът и квадратът съдържат безброй много точки, вероятността p за успех се пресмята като отношение на лицата (геометрична вероятност):



$$p = \frac{S_{\text{четвърт кръг}}}{S_{\text{квадрат}}} = \frac{\pi/4}{1} = \frac{\pi}{4}.$$

Следователно $\xi \sim \text{Bi}(n,\ p)$. По формулите за биномното разпределение пресмятаме:

$$\mathbf{D}\xi = np(1-p) = 30000 \cdot \frac{\pi}{4} \cdot \left(1 - \frac{\pi}{4}\right) = 7500\,\pi\left(1 - \frac{\pi}{4}\right);$$

$$\sigma(\xi) = \sqrt{\mathbf{D}\xi} = \sqrt{7500\,\pi\left(1 - \frac{\pi}{4}\right)}\,; \qquad \mathbf{E}\xi = np = 30000 \cdot \frac{\pi}{4} = 7500\,\pi.$$

Програмният код отпечатва обаче стойността на `count`, разделена на 7500. С помощта на свойствата на математическото очакване и стандартното отклонение намираме:

$$\mathbf{E}\left(\frac{\xi}{7500}\right) = \frac{\mathbf{E}\xi}{7500} = \frac{7500\,\pi}{7500} = \pi,$$ т.е. отпечатаната стойност е около $\pi \approx 3{,}14$;

$$\sigma\left(\frac{\xi}{7500}\right) = \frac{\sigma(\xi)}{7500} = \sqrt{\frac{\pi}{7500}\left(1 - \frac{\pi}{4}\right)} \approx 0{,}009 \approx 0{,}01$$ е стандартното отклонение на отпечатаната.стойност. Действително, експериментите показват, че тя най-често е в интервала между 3,13 и 3,15.

Съпоставка между двата типа рандомизирани алгоритми

По-горе дефинирахме два типа рандомизирани алгоритми:
— алгоритми от тип Лас Вегас — резултатът винаги е коректен, но времето за работа е случайна величина, която може да бъде неограничена (отгоре), т.е. такъв алгоритъм може да работи безкрайно;
— алгоритми от тип Монте Карло — резултатът понякога е грешен; за сметка на това времето за работа (което може да бъде случайна величина) е ограничено отгоре от детерминирана функция на дължината на входа.

Пример 8: Даден е масив $M[1...n]$, състоящ се от равен брой знаци ‘a’ и ‘b’, като дължината на масива n е четно число. Търси се индексът на произволен знак ‘a’. Разглеждаме два рандомизирани алгоритъма:

```
findA_LasVegas(M[1...n])
repeat
   i ← rand(1...n)
until M[i] = 'a'
return i
```

Този алгоритъм е от тип Лас Вегас, защото винаги връща правилен резултат. Но времето за работа е неограничена отгоре случайна величина: теоретично такъв алгоритъм може да работи безкрайно, макар че вероятността това да се случи е безкрайно малка. По-точно, величината ξ = брой итерации (която е равна по порядък на времето за изпълнение) има геометрично разпределение от единица, като вероятността за успех (т.е. вероятността за срещане на знак ‘a’) е една и съща при всяка проверка: $p = 1/2$. Затова математическото очакване на времето за изпълнение е крайно: $\mathbf{E}\xi = \frac{1}{p} = \frac{1}{1/2} = 2$, т.е. знак ‘a’ ще бъде открит средно от втория опит. Понеже 2 е константа (не зависи от n), то очакваното време за работа на алгоритъма е $\Theta(1)$.

Усредняването на времето на алгоритъма е по възможните случайни изходи на `rand`, а не по входните данни. По входните данни се разглежда, както обикновено, най-лошият възможен случай. Но в конкретния пример всички възможни входове са еднакво лоши. Това е едно предимство на рандомизираните алгоритми: те не могат да бъдат атакувани лесно от злонамерен противник.

```
findA_MonteCarlo(M[1...n])
for k ← 1 to 10
   i ← rand(1...n)
   if M[i] = 'a'
      return i
return -1
```

Този алгоритъм е от тип Монте Карло, защото неговото време за работа (макар и случайна величина) е ограничено отгоре: правят се най-много 10 опита за откриване на знак ‘a’. Детерминираната функция 10 не зависи от n, т.е. времето за работа е константа: $\Theta(1)$. За сметка на това има известна вероятност за неуспешно приключване на алгоритъма: той ще върне –1 (знак за провал), ако и десетте опита са неуспешни. Вероятността за неуспех е еднаква $(1/2)$ при всички опити и опитите са независими. От теоремата за умножение на вероятности намираме, че вероятността за провал на алгоритъма (тоест вероятността за десет неуспешни проверки) е равна на $(1/2)^{10} = 1/1024$, т.е. по-малка от едно на хиляда.

Превръщане на рандомизиран алгоритъм от един тип в друг

Алгоритъм от тип Лас Вегас може винаги да бъде превърнат в алгоритъм от тип Монте Карло: едновременно с оригиналния алгоритъм се пуска таймер; след изтичането на предварително определено време, ако алгоритъмът все още не е приключил работа, го спираме аварийно и връщаме индикатор за провал. На практика вместо ограничение на реалното време можем да наложим ограничение на броя на итерациите на подходящо избран цикъл. Броят на итерациите се избира тъй, че новият алгоритъм (от тип Монте Карло) да има достатъчно висока, предварително определена вероятност за правилен резултат. Необходимият брой итерации може да бъде намерен по зададена доверителна вероятност с помощта на свойствата на вероятността от първия раздел на урока.

Задача 3. Алгоритъмът от тип Монте Карло, разгледан в пример 8, изпълняваше десет итерации на цикъла и допускаше вероятност за грешен резултат под едно на хиляда. Колко итерации са нужни, та вероятността за грешен отговор да спадне под едно на милион?

Решение: Алгоритъмът връща грешен отговор само тогава, когато всички опити са неуспешни. Понеже опитите са независими, то вероятността за грешен резултат общо от всички опити е равна на произведението на отделните вероятности, т.е. $\left(\frac{1}{2}\right)^k$, където k е броят на опитите (т.е. броят на итерациите). Решаваме неравенството $\left(\frac{1}{2}\right)^k < \frac{1}{1000000}$ и получаваме $k > \log_2 1000000 \approx 19{,}93$. Понеже k е цяло число, то $k \geq 20$, т.е. нужни са 20 опита / итерации, та вероятността за неуспешно търсене да спадне под едно на милион.

Алгоритъм от тип Монте Карло може в повечето случаи (не винаги!) да бъде превърнат в алгоритъм от тип Лас Вегас. За целта оригиналният алгоритъм се изпълнява многократно, като след всяко изпълнение се прави проверка на получения резултат. Към ново изпълнение се пристъпва само ако резултатът от последното изпълнение е грешен. С други думи, алгоритъмът от тип Монте Карло се пуска отново и отново, докато върне верен резултат. Проблемът е дали съществува (проста) процедура за проверка на върнатия резултат.

В пример 8 такава процедура има: резултатът е верен, ако и само ако е различен от -1, тъй като тази стойност е сигнал за неуспешно търсене.

В дефиницията на алгоритъм от тип Монте Карло няма изискване алгоритъмът да връща специална стойност при неуспех. Алгоритъмът от пример 8 би могъл да връща 1 вместо -1, когато не успее да намери знак ‘a’. Сега от резултата j няма да е очевидно дали е верен, но проста процедура за проверка все пак има: достатъчно е да проверим дали M [j] = ‘a’.

В други задачи може да няма процедура за проверка. Например не е ясно как можем (бързо) да проверим дали шахматен ход, предложен като най-добър, наистина е такъв.

Бележки по класа BPP

1) В класа **BPP** по дефиниция се допускат само алгоритми, чиято вероятност за грешен отговор (yes или no) е по-малка от $\frac{1}{3}$. Това число с нищо не е специално и може да бъде заменено с произволна стойност (между 0 и $\frac{1}{2}$). Колкото и малка да е вероятността за грешка, тя може да бъде постигната, като оригиналният алгоритъм се пусне няколко пъти (нечетен брой) и за резултат се вземе по-често връщаната стойност. Ако алгоритъмът греши поначало в $\frac{1}{3}$ от отговорите си, то с помощта на биномното разпределение можем да пресметнем, че например при 23 повторения вероятността за грешка ще спадне под 5 %.

2) Очевидно **P** $\subseteq$ **BPP**. Засега не се знае дали това включване е строго.