1. Wstęp. Program ten jest jednym z rozwiązań następującego problemu:

> Na wejściu dane są dwie liczby naturalne M i N takie, że $M \leqslant N$. Na wyjściu otrzymujemy M-elementowy, rosnący ciąg losowo wybranych liczb z przedziału $[1\,..\,N]$.

Problem ten wraz z rozwiązaniem pojawił się w kolumnie „Programming Pearls" [CACM 1984]. Według D. E. Knutha, autora rozwiązania, poniższe podejście prowadzi do najefektywniejszego programu, gdy M jest dość duże i jednocześnie małe względem N.

Program ten można wykorzystać do generowania „szóstek" w TOTO-LOTKA. Liczby do zaznaczenia na kuponie otrzymamy uruchamiając go w następujący sposób: `sample 6 49`.

Wygenerowany ciąg liczb losowych zależy od liczby sekund, które upłynęły od dnia 1 stycznia 1970 roku. I tak, po uruchomieniu programu dnia 24 września 1996 roku o godzinie 13:05:53 (czas podany z zegara komputera), zostały wygenerowane następujące liczby: `5 19 24 29 30 31`.

2. Zmienne M i N to dane programu, opisane powyżej. Oto ich deklaracje.

```
#define M_max 1001    /* maksymalna wielkość M dozwolona w tym programie */
```

⟨ Zmienne globalne 2 ⟩ ≡

```
int M;    /* liczba elementów w próbce */
int N;    /* losujemy liczby z przedziału [1, N] */
```

Kontynuacja: sekcje 7, 11 i 14.

Ten kod jest użyty w sekcji 6.

3. Funkcje generujące liczby losowe są zadeklarowane w pliku `<stdlib.h>`. Tam też zdefiniowano stałą `RAND_MAX`. Do inicjalizacji generatora liczb losowych wykorzystamy liczbę sekund zwracaną przez funkcję *time*.

```
#include <stdio.h>
#include <stdlib.h>    /* deklaracje funkcji exit, rand i srand */
#include <time.h>      /* deklaracja funkcji time */
```

4. Będziemy też potrzebować funkcji *liczba_losowa*(i) wybierającej losowo liczbę z przedziału $[1\,..\,i]$. Strona podręcznika dotycząca funkcji *rand* zaleca aby użyć poniższego wyrażenia.

```
unsigned int liczba_losowa(i)
    unsigned int i;
{
  return 1 + (unsigned int)((float) i * rand( )/(RAND_MAX + 1.0));
}
```

5. ⟨ Zainicjalizuj generator liczb losowych 5 ⟩ ≡

```
srand((unsigned int) time((time_t) Λ));
```

Ten kod jest użyty w sekcji 6.

6. Ogólny zarys programu. Po wczytaniu liczb M i N z linii poleceń i inicjalizacji generatora liczb losowych, generujemy kolejno elementy ciągu w pętli **while**.

```
⟨ Zmienne globalne 2 ⟩
int main(argc, argv)
    int argc;      /* liczba argumentów w linii poleceń */
    char *argv[];      /* argumenty */
{
  ⟨ Wczytaj wartości M i N z linii poleceń 8 ⟩
  ⟨ Zainicjalizuj generator liczb losowych 5 ⟩
  ⟨ Zacznij od zbioru pustego S 10 ⟩
  liczba_elementów = 0;
  while (liczba_elementów < M) {
    t = liczba_losowa(N);
    ⟨ Jeśli t nie należy do S, to wstaw t i zwiększ liczbę_elementów 12 ⟩
  }
  ⟨ Wypisz elementy S w porządku rosnącym 15 ⟩
  return EXIT_SUCCESS;
}
```

7. W schemacie opisanym powyżej pojawiło się kilka nowych zmiennych, zadeklarujmy je teraz. Reprezentacja zbioru S zostanie wprowadzona w sekcji 11.

```
⟨ Zmienne globalne 2 ⟩ +≡
  unsigned int liczba_elementów;      /* liczba elementów zbioru S */
  unsigned int t;      /* nowy kandydat na element zbioru S */
```

8.
```
⟨ Wczytaj wartości M i N z linii poleceń 8 ⟩ ≡
  switch (argc) {
  case 3:
    if (sscanf((char *) argv[1], "%d", &M) ≡ 1 ∧ sscanf((char *) argv[2], "%d", &N) ≡ 1) break;
  default: wypisz_sposób_użycia: fprintf(stderr,
        "Użycie: %s M N\n  gdzie M należy do przedziału [1..N] i 1<=N<=%d\n", *argv, M_max);
    exit(−1);
  }
  ⟨ Sprawdź poprawność danych wejściowych 9 ⟩
```

Ten kod jest użyty w sekcji 6.

9.
```
⟨ Sprawdź poprawność danych wejściowych 9 ⟩ ≡
  if (N < 1 ∨ M < 1 ∨ M > N ∨ M > M_max ∨ N > M_max) {
    fprintf(stderr, "! Należy podać liczby M, N takie, że M<=N.\n");
    goto wypisz_sposób_użycia;
  }
```

Ten kod jest użyty w sekcji 8.

10.
```
⟨ Zacznij od zbioru pustego S 10 ⟩ ≡
  H_max = 2 * M − 1;
  alpha = (2.0 * M)/N;
  for (h = 0; h ≤ H_max; h++) hash[h] = 0;
```

Ten kod jest użyty w sekcji 6.

11. Uporządkowane tablice rozproszone. Kluczowym pomysłem prowadzącym do sprawnie działającego programu jest tworzenie zbioru S w taki sposób, aby jego elementy można było łatwo i szybko uporządkować. Jak zobaczymy, metoda „uporządkowanych tablic rozproszonych” [Amble i Knuth, *The Computer Journal* **17**, 135–142] idealnie nadaje się do tego zadania.

Tablice rozproszone posiadają następującą własność: *Elementy uporządkowanej tablicy rozproszonej nie zależą od kolejności w której były wstawiane.* Dlatego uporządkowana tablica rozproszona stanowi „kanoniczną” reprezentację zbioru jej elementów.

Zbiór S będzie reprezentowany przez tablicę składającą się z $2M$ liczb całkowitych.

```
⟨ Zmienne globalne 2 ⟩ +≡
  unsigned int hash[2 * M_max];      /* uporządkowana tablica rozproszona */
  unsigned int h;      /* indeks w hash */
  unsigned int H_max;      /* maksymalna liczba elementów do wstawienia w hash */
  float alpha;      /* rozmiar tablicy/N */
```

12. Tutaj zajmiemy się umieszczeniem elementu t w tablicy rozproszonej. W tym celu użyjemy funkcji mieszającej obliczającej adres h według wzoru:

$$h = \lfloor 2M(t-1)/N \rfloor .$$

Zauważmy, że jest to funkcja rosnąca względem t o prawie jednostajnym rozkładzie w przedziale $0 \leqslant h < 2M$.

```
⟨ Jeśli t nie należy do S, to wstaw t i zwiększ liczbę_elementów 12 ⟩ ≡
  h = (int) alpha * (t - 1);
  while (hash[h] > t)
    if (h ≡ 0) h = H_max; else h−−;
  if (hash[h] < t) {      /* t nie występuje w S */
    liczba_elementów++;
    ⟨ Wstaw t w uporządkowaną tablicę rozproszoną 13 ⟩
  }
```

Ten kod jest użyty w sekcji 6.

13. Najważniejszą część algorytmu stanowi metoda wstawiania elementu do tablicy rozproszonej. Nowy element t jest wstawiany w miejsce poprzedniego elementu $t_1 < t$, który zostanie wstawiony w miejsce $t_2 < t_1$ itd., aż do znalezienia wolnego miejsca.

```
⟨ Wstaw t w uporządkowaną tablicę rozproszoną 13 ⟩ ≡
  while (hash[h] > 0) {
    t_1 = hash[h];      /* mamy 0 < t_1 < t */
    hash[h] = t;
    t = t_1;
    do {
      if (h ≡ 0) h = H_max; else h−−;
    } while (hash[h] ≥ t);
  }
  hash[h] = t;
```

Ten kod jest użyty w sekcji 12.

14.
```
⟨ Zmienne globalne 2 ⟩ +≡
  unsigned int t_1;      /* przesuwany element tablicy */
```

15. Sortowanie w czasie liniowym. Kulminacyjnym fragmentem tego programu jest możliwość prostego odczytania w porządku rosnącym elementów z tablicy rozproszonej. Dlaczego jest to możliwe? Jak to powiedziano, ostateczny stan tablicy nie zależy od kolejności, w której wstawiano elementy. Ponieważ używamy monotonicznej funkcji mieszającej, łatwo można sobie wyobrazić, jak tablica się zmienia w przypadku wstawiania elementów od największego do najmniejszego.

Przyjmijmy, że niezerowe elementy w tablicy *hash*, to $T_1 < \cdots < T_M$. Jeśli k z nich, w trakcie wstawiania, przesunięto z początku tablicy na jej koniec (tj. kiedy h zmieniało wartość z 0 na *H_max* k razy), to $hash[0]$ będzie zawierać zero (w tym przypadku k też musi być równe zero), albo będzie zawierać T_{k+1}. W tym przypadku, elementy $T_{k+1} < \cdots < T_M$ i $T_1 < \cdots < T_k$ pojawią się w tablicy uporządkowane od lewej do prawej. Dlatego wstawione elementy można wypisać w porządku rosnącym w wyniku dwukrotnego przejrzenia tablicy!

```
#define wypisz_element printf("%u␣",hash[h])
#define wypisz_nl printf("\n")
```

⟨ Wypisz elementy S w porządku rosnącym 15 ⟩ ≡

```
if (hash[0] ≡ 0) {      /* nie było przesunięcia na pozycję H_max */
  for (h = 1; h ≤ H_max; h++)
    if (hash[h] > 0) wypisz_element;
}
else {      /* było przesunięcie na pozycję H_max */
  for (h = 1; h ≤ H_max; h++)
    if (hash[h] > 0)
      if (hash[h] < hash[0]) wypisz_element;
  for (h = 0; h ≤ H_max; h++)
    if (hash[h] ≥ hash[0]) wypisz_element;
}
wypisz_nl;
```

Ten kod jest użyty w sekcji 6.

16. Skorowidz. Poniżej znajdziesz listę identyfikatorów użytych w programie `hello.w`. Liczba wskazuje na numer sekcji, w której użyto identyfikatora, a liczba podkreślona — numer sekcji w której zdefiniowano identyfikator.

⟨ Jeśli t nie należy do S, to wstaw t i zwiększ *liczbę_elementów* 12 ⟩ Użyto w sekcji 6.
⟨ Sprawdź poprawność danych wejściowych 9 ⟩ Użyto w sekcji 8.
⟨ Wczytaj wartości M i N z linii poleceń 8 ⟩ Użyto w sekcji 6.
⟨ Wstaw t w uporządkowaną tablicę rozproszoną 13 ⟩ Użyto w sekcji 12.
⟨ Wypisz elementy S w porządku rosnącym 15 ⟩ Użyto w sekcji 6.
⟨ Zacznij od zbioru pustego S 10 ⟩ Użyto w sekcji 6.
⟨ Zainicjalizuj generator liczb losowych 5 ⟩ Użyto w sekcji 6.
⟨ Zmienne globalne 2, 7, 11, 14 ⟩ Użyto w sekcji 6.

PRÓBKA LOSOWA

przykład programu opisowego

(wersja 1.1.1.1)

Program ten generuje ciąg przypadkowych liczb naturalnych uporządkowanych rosnąco. Jest to wersja programu, który po raz pierwszy pojawił się w czasopiśmie CACM w 1984 roku.

`/var/cvs/literate/examples/wbzyl/sample.w,v` `2003/09/24`