ДИНАМИЧНО ПРОГРАМИРАНЕ И АЛЧНИ АЛГОРИТМИ
ДОМАШНО № 3 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 1. ПОТОК
(СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2019 / 2020 УЧ. Г.)

Задача 1. Образуваме числови редици по следното правило: първият член е числото 1, всеки следващ член се получава с прибавяне на единица или умножение по две, или по три. Да се намери най-късата редица от операции, чрез която от числото 1 се получава числото n. Навсякъде в задачата n е цяло положително число.

Съставете алгоритъм, решаващ тази задача за време $O(n)$ и с допълнителна памет $O(n)$. Алгоритъмът получава на стандартния вход цели положителни числа, по едно на ред, и отпечатва на всеки ред от стандартния изход съответната най-къса редица от операции. Ако има няколко най-къси редици от операции, алгоритъмът отпечатва само една от тях. Всяка операция се представя със своя код:

1 — прибавяне на единица;
2 — умножение по две;
3 — умножение по три.

Алгоритъмът да се опише на езика Си само с типовете цяло число и масив от цели числа. Да не се използват готови функции освен тези за вход-изход. Числото 0 е край на входа.

Пример: При $n = 6$ възможните редици от операции са "23", "32", "13", "112" и "11111". Първите три са най-къси и алгоритъмът трябва да отпечата една от тях (няма значение коя) без кавичките. При $n = 1$ алгоритъмът трябва да отпечата празен ред. **(5 точки)**

Задача 2. Разглеждаме следния ***алчен алгоритъм*** за решаването на задача 1:

1) Ако даденото число n се дели на 3, делим го на 3.
2) В противен случай, ако n се дели на 2, делим го на 2.
3) В противен случай изваждаме единица от n.
4) Спираме, когато n стане равно на единица.
5) Най-късата редица от операции се състои от обратните действия в обратен ред.

Програмен код на Си:

```
void ALG(unsigned int n) { // n > 0
  if (n % 3 == 0) {
    ALG(n/3);
    printf("3");
  }
  else if (n % 2 == 0) {
    ALG(n/2);
    printf("2");
  }
  else if (n > 1) {
    ALG(n-1);
    printf("1");
  }
}
```

а) Вярно ли е, че ALG намира най-къса редица от операции за всяко n?

Ако да — докажете това с подробни разсъждения.

Ако не — предложете n, опровергаващо твърдението, и обяснете в какво се състои опровержението.

(1 точка)

б) Времето за работа на ALG мерим с дължината $T(n)$ на получената редица. Намерете порядъка на функцията $T(n)$.

(4 точки)

РЕШЕНИЯ

Задача 1 се решава с помощта на ***динамично програмиране:***

```
typedef unsigned int natural;

void Relax(natural n, natural k, natural r, natural op,
           natural * dyn, natural * prv, natural * opr) {
  if (r > n) return;
  if (dyn[r] <= dyn[k] + 1) return;
  dyn[r] = dyn[k] + 1;  prv[r] = k;  opr[r] = op;
}

void FindShortestSequence(natural n) {   // n > 0
  natural k, p;
  natural * dyn, * prv, * opr;
  if (n < 1) return;
  if (n == 1) { printf("\n"); return; }
  dyn = (natural *) malloc(  n   * sizeof(natural));
  prv = (natural *) malloc((n-1) * sizeof(natural));
  opr = (natural *) malloc((n-1) * sizeof(natural));
  dyn = &(dyn[-1]); prv = &(prv[-2]); opr = &(opr[-2]);
  dyn[1] = 0;
  for (k = 2; k <= n; ++k) dyn[k] = n; // +infinity
  for (k = 1; k < n; ++k)
    for (p = 1; p <= 3; ++p)
      Relax(n, k, ((p == 1) ? (1 + k) : (p * k)), p, dyn, prv, opr);
  k = p = n;
  while (k > 1) { opr[p--] = opr[k]; k = prv[k]; }
  for (++p; p <= n; ++p) printf("%u", opr[p]);
  printf("\n");
  dyn = &(dyn[+1]); prv = &(prv[+2]); opr = &(opr[+2]);
  free(dyn); free(prv); free(opr);
}

int main(int argc, char * argv[]) {
  natural n = 1;
  while (n > 0) { scanf("%u", &n); FindShortestSequence(n); }
  return 0;
}
```

Задача 2. Алчният алгоритъм не е коректен. Контрапример: при $n = 10$ алчният алгоритъм връща редицата "2212", но има по-къса редица от операции: "331".

Дължината $T(n)$ на редицата от операции, отпечатана от алгоритъма `ALG`, се задава с началното условие $T(1) = 0$ и рекурентното уравнение

$$T(n) = \begin{cases} T(n / 3) + 1, & \text{ако } n \text{ се дели на 3;} \\ T(n / 2) + 1, & \text{ако } n \text{ се дели на 2, но не и на 3;} \\ T(n - 1) + 1, & \text{ако } n \text{ не се дели нито на 2, нито на 3.} \end{cases}$$

Не е лесно, а и не е нужно да решим това уравнение в явен вид. Достатъчно е да получим оценка отгоре и отдолу на порядъка на $T(n)$, а той зависи от скоростта, с която намалява аргументът на функцията. За съжаление, скоростите в трите случая са твърде различни: делението води до геометрична прогресия, а изваждането — до аритметична. Следователно горната и долната граница на $T(n)$ ще имат различен порядък, ако използваме тези формули.

Ще преодолеем възникналата трудност, като развием третия случай само с една стъпка. Числото n не се дели нито на 2, нито на 3 $\Leftrightarrow$ n дава остатък 1 или 5 при деление на 6. Тези два подслучая разглеждаме поотделно.

Ако n дава остатък 1 при деление на 6, то $n - 1$ се дели на 6, а оттам се дели и на 3. Тогава $T(n - 1) = T[(n - 1) / 3] + 1$, следователно $T(n) = T[(n - 1) / 3] + 2$.

Ако пък n дава остатък 5 при деление на 6, то $n - 1$ се дели на 2, но не се дели на 3. Тогава $T(n - 1) = T[(n - 1) / 2] + 1$, следователно $T(n) = T[(n - 1) / 2] + 2$.

Рекурентното уравнение приема следния вид:

$$T(n) = \begin{cases} T(n / 3) + 1, & \text{ако } n \text{ дава остатък 0 или 3 при деление на 6;} \\ T(n / 2) + 1, & \text{ако } n \text{ дава остатък 2 или 4 при деление на 6;} \\ T(n / 3) + 2, & \text{ако } n \text{ дава остатък 1 при деление на 6;} \\ T(n / 2) + 2, & \text{ако } n \text{ дава остатък 5 при деление на 6.} \end{cases}$$

Аргументите на функцията са винаги цели числа. В последните два случая се подразбира, както обикновено, че делението е със закръгляне надолу за сметка на пропуснатата операция изваждане на единица.

Новите четири формули приличат на уравненията от тип "разделяй и владей". Знаем, че те се решават с мастър-теоремата и че асимптотичният порядък на решението не зависи от началното условие (то определя само константния множител пред порядъка).

Да разгледаме четирите функции $T_{b,\,c}(n)$, определени с подходящи начални условия и с рекурентното уравнение

$$T_{b,c}(n) = T_{b,c}(n / b) + c,$$

важащо за достатъчно големи n. Параметрите b и c са константи, тоест не зависят от n. По-точно, b е 2 или 3, а пък c е 1 или 2. От мастър-теоремата следва, че четирите функции имат един и същи асимптотичен порядък:

$$T_{b,c}(n) = \Theta(\log n).$$

При функцията $T(n)$ параметрите b и c се променят в зависимост от n, с което се променя скоростта на нарастване на функцията. Следователно асимптотичният порядък на $T(n)$ е междинен спрямо порядъка на най-бавно и най-бързо растящата от четирите функции. Но техният порядък на нарастване е един и същ (може да има разлики в множителите), така че той е също порядък на нарастване на функцията $T(n)$:

$$T(n) = \Theta(\log n).$$

С други думи, броят на операциите, чрез които алчният алгоритъм получава дадено число, започвайки от единицата, има логаритмичен порядък спрямо това число.