Оптимизация использования памяти: битовые массивы и линеаризация многомерных массивов

Е.Э. Темиргалеев

В заметке приведена пара методов, позволяющих для определённых задач оптимизировать объём используемой памяти. Применимы для многих современных императивных ЯП. В качестве нотации используется ЯВУ оберон-семейства Компонентный Паскаль (КП). Большинство примеров приведено на КП для его реализации в системе Блэкбокс.

Битовые массивы. Применение массива битов для представления массивов, типом элементов которых является двухэлементное множество, с точки зрения задействуемого объёма памяти оптимально для двоичных компьютеров. Например, для классического двухэлементного множества, представленного в КП типом $BOOLEAN$, представление $p:\ ARRAY\ m\ OF\ BOOLEAN$ битовым массивом сократит объём приблизительно в восемь раз ($SIZE($ $BOOLEAN) = 1$ (Байт)).

Битовые массивы представлены в языках оберон-семейства типом SET (множество небольших целых чисел $\{0, \ldots, n-1\}$, суть битовый массив размерности n) [1]. В КП $n = MAX(SET)+1$ и массив p может быть представлен как $q:\ ARRAY\ RoundUp(m/n)\ OF\ SET$ (или просто SET, если $m \leq n$). $RoundUp$ — функция округления до ближайшего целого в сторону положительной бесконечности, которая может быть выражена как $(m+(n-1))DIVn$. Запись и чтение элементов будут представлены:

```
ASSERT((0 <= i) & (i < m)); (* i = 0..m-1 *)
EXCL(q[i DIV n], i MOD n) (* p[i] := FALSE *)
INCL(q[i DIV n], i MOD n) (* p[i] := TRUE *)
(i MOD n) IN q[i DIV n]) = p[i]
```

В некоторых случаях применение SET может не только сократить объём памяти, используемой под данные, но и под код, а также исходный текст программы, делая его более ясным. См. пример ниже, также [2] — пример замены $ARRAY\ 18\ OF\ BOOLEAN$ на SET.

Линеаризация многомерных массивов. *Линеаризация* многомерного массива — представление его в виде одномерного (линейного). Например, линеаризация двумерного массива $p:\ ARRAY\ 2,3\ OF\ T$

$$p[0,0], p[0,1], p[0,2]$$
$$p[1,0], p[1,1], p[1,2]$$

по строкам

$$p[0,0], p[0,1], p[0,2], p[1,0], p[1,1], p[1,2]$$

будет представлена массивом $l:\ ARRAY\ 2*3\ OF\ T$

$$l[0], l[1], l[2], l[3], l[4], l[5]$$

где $p[y,x] = l[3y+x], 0 \le y < 2, 0 \le x < 3$

Для $p:\ ARRAY\ x_{k-1}, ..., x_1, x_0\ OF\ T$,

$$l:\ ARRAY\ x_{k-1} * \cdots * x_0\ OF\ T$$
$$p[i_{k-1}, \ldots, i_1, i_0] = l[(x_{k-2} \ldots x_1 x_0) i_{k-1} + \cdots + x_0 i_1 + i_0]$$

Для случаев, когда представление одномерного массива выходит более компактным, нежели представление многомерного, линеаризация является методом экономии памяти. Например, для представления $p:\ ARRAY\ 4,5\ OF\ BOOLEAN$ может использоваться $q_1:\ ARRAY\ 4\ OF\ SET$, в то время как для представления $l:\ ARRAY\ 4*5\ OF\ BOOLEAN$ достаточно $q_2:\ SET$.

Другим примером применения линеаризации является организация многомерных динамических массивов в ЯП, поддерживающих только одномерные. Довольно часто сейчас можно встретить упоминание метода формирования двумерного массива r строк на c столбцов в виде $r+1$ одномерных массивов — r массивов-строк размерности c и массива указателей на строки размерности r. Например [3, «Пример динамического массива на C++»] (комментарии и пример одномерного случая опущены; $r = 16, c = 8$):

```
int **array2;
array2 = new int*[16];
for(int i = 0; i < 16; i++)
  array2[i] = new int[8];
```

Линеаризованное представление по сравнению с данным является более компактным за счёт отсутствия массива указателей и меньшего числа блоков динамической памяти.

Пример: представление двумерного массива целых $\{0,1\}$ одномерным битовым массивом. Фигура игры тетрис определяется набором фаз, каждая из которых представляет отдельное положение фигуры при её поворотах [4]. Фаза представлена наиболее простым с методической точки зрения способом — двумерным массивом целых $\{0,1\}$.

Модуль *TetrisFigures* [5], реализующий игровые фигуры, рассчитан на фигуры произвольного размера, поэтому фаза — динамический массив.

```
TYPE
  Phase = POINTER TO ARRAY OF ARRAY OF BYTE;
  Figure = POINTER TO LIMITED RECORD
    w-, h-: INTEGER;
```

```
    ...
    (f: Figure) AddPhase (phase: Phase), NEW;
    (f: Figure) NextPhase, NEW;
    (f: Figure) Phase (): Phase, NEW;
    ...
  END;
```

Пример процедуры формирования фазы:

```
(* фигура 1, 3x3, "треугольник", фаза 2. *)
PROCEDURE Fig1p2* (): Phase;
  VAR p: Phase;
BEGIN
  NEW(p, 3, 3);
  p[0,0] := 0; p[0,1] := 1; p[0,2] := 0;
  p[1,0] := 0; p[1,1] := 1; p[1,2] := 1;
  p[2,0] := 0; p[2,1] := 1; p[2,2] := 0;
  RETURN p
END Fig1p2;
```

Представление фазы — предмет для двух описанных выше способов оптимизации:

- использование битового массива вместо массива целых $\{0, 1\}$;
- линеаризации двумерного массива с целью задействовать все биты SET.

```
TYPE Phase = POINTER TO ARRAY OF BYTE;
PROCEDURE Fig1p2* (): Phase;
  VAR l: Phase;
BEGIN
  NEW(l, 9);  (* 9 = 3*3 *)
  l[0] := 0; l[1] := 1; l[2] := 0;  (* 3*0 + i, i=0..2 *)
  l[3] := 0; l[4] := 1; l[5] := 1;  (* 3*1 + i, i=0..2 *)
  l[6] := 0; l[7] := 1; l[8] := 0;  (* 3*2 + i, i=0..2 *)
  RETURN l
END Fig1p2;
```

```
TYPE Phase = POINTER TO ARRAY OF SET;
PROCEDURE Fig1p2* (): Phase;
  VAR q: Phase;
BEGIN
  NEW(q, 1);  (* 1 = (9 + 31) DIV 32 *)
  q[0] := {1, 4..5, 7};
  RETURN q
END Fig1p2;
```

Чтение элементов $Phase$ фигуры f в процедурах $TetrisModels.CheckFigure$ и $TetrisModels.MaskFigure$, выполняемое последовательно по схеме:

```
phase := f.Phase();
y := 0;
FOR row := ... DO  (* f.h итераций -> 0 <= y < f.h *)
  x := 0;
  FOR col := ... DO  (* f.w итераций -> 0 <= x < f.w *)
    IF (phase[y, x] = 1) ...
    INC(x)
  END;
  INC(y)
END
```

изменится на:

```
phase := f.Phase();
i := 0;  (* 0 <= i < f.h * f.w *)
FOR row := ... DO  (* f.h итераций *)
  FOR col := ... DO  (* f.w итераций *)
    IF (i MOD 32) IN phase[i DIV 32]) ...
    INC(i)
  END
END
```

Список литературы

[1] Вирт Н. SET: Недооцениваемый тип данных и его компиляция для ARM (дата обновления: 07.06.2012). // URL: `http://oberoncore.ru/library/wirth_sets` (дата обращения: 11.06.2012).

[2] Темиргалеев Е. Э. Использование SET вместо ARRAY n OF BOOLEAN в CpcBeautifierOptions (дата обновления 07.06.2012). // URL: `http://oberoncore.ru/wiki/blackbox/ex/cpcbeautifieroptions-array_of_boolean_as_set` (дата обращения: 11.06.2012).

[3] Индексный массив: Википедия, свободная энциклопедия. Версия от 24.02.2012. // URL: `http://ru.wikipedia.org/wiki/Индексный_массив` (дата обращения: 11.06.2012).

[4] Кузьмицкий И. А. Создаём тетрис в среде BlackBox Component Builder (дата обновления: 27.04.2009). // URL: `http://oberoncore.ru/library/kuzmiczkij_sozdayom_tetris_v_srede_blackbox` (дата обращения: 11.06.2012)

[5] Кузьмицкий И. А. Подсистема Tetris (дата обновления: 01.05.2012). // URL: http://oberoncore.ru/bbcc/subs/tetris/ (дата обращения: 11.06.2012)