**СПЕЦИФИКАЦИЯ БАЙТ-КОДА ДЛЯ ИССЛЕДОВАНИЯ ПРИМЕНИМОСТИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ГЕНЕРАЦИИ ПРОГРАММНОГО КОДА РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА**

**Cтроение памяти**

Память представлена стеком, реализованным в виде двух массивов:

- МассивТипов - Массив из байтов типа (таблица 1);

- МассивУказателей - Массив из sizeof(intptr\_t)-байтовых элементов (элемент этого массива по индексу либо число, либо указатель на память в куче, в зависимости от того, чему равен байт типа по индексу).

Дополнительно предоставляется память в виде глобальных переменных (таблица 2).

**Типизация**

Байт-код предоставляет только 5 базовых типов: число, массив чисел, указатели на то и другое и функцию (таблица 1). Типизация строгая, явная и статическая: нельзя интерпретировать число как указатель на массив или наоборот. Данное правило ожидаемо распространяется и на указатели, хотя байт-кодом определены команды, которые на вход получают обобщенный указатель и работают с ним в зависимости от того, на что он указывает (команда print в таблице 1 приложения А). Байт-код не предоставляет средств для создания пользовательских типов ввиду узконаправленности байт-кода.

Функция – тип, аналогичный указателю на функцию; переменная типа указателя на функцию хранит индекс команды function в тексте программы и создаётся при определении функции в тексте программы. Все функции байт-кода являются процедурами (не возвращают значений), но, как и в языке С, получить результат функции все ещё можно через аргумент-указатель. Так как в языке не предусмотрено разыменование указателей, с указателями работают лишь специализированные команды самого языка, то передача указателей в функции производится через промежуточный ссылочный тип.

Таблица 1 - Типы данных

|  |  |  |
| --- | --- | --- |
| **Название типа далее** | **Название типа на с++** | **Описание** |
| int | intptr\_t | Число |
| int\* | intptr\_t | Индекс числа в МассивУказателей |
| arr\_int | std::vector<intptr\_t> | Массив |
| arr\_int\* | intptr\_t | Индекс массива в МассивУказателей |
| arr\_arr\_int | std::vector<std::vector<intptr\_t>> | Двумерный массив |
| arr\_arr\_int\* | intptr\_t | Индекс двумерного массива в МассивУказателей |
| function | intptr\_t | Адрес функции в исполняемом коде |

Байт-код имеет строгий список входных и выходных значений. Входные – это исходный граф (квадратная матрица смежности) и длина и ширина матрицы. Выходное – массив-результат result из вершин, по которым проходит найденный кратчайший путь.

Память под result выделяется перед началом интерпретации байт-кода, поэтому в стеке он всегда лежит по индексу 0. Result имеет длину size и по умолчанию заполняется числами с 0 до size-1 – то есть, содержит список всех вершин. Таким образом, для size = 5, result = { 0, 1, 2, 3, 4 }. Так как особь, содержащая байт-код, может в результате работы ГА сразу вернуть result, созданный по умолчанию, либо его перевернутую версию, то процессор байт-кода следит за тем, чтобы result как-то изменялся.

Таблица 2 - Глобальные переменные кода

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип представления** | **Описание** |
| graph | const arr\_arr\_int | Квадратная матрица смежности исходного графа. |
| size | const int | Ширина и длина матрицы. |

**2.3.3 Обзор команд**

Список команд байт-кода (таблица 1 приложения А) содержит и классические команды языков ассемблера, и более высокоуровневые операторы. Например, байт-код поддерживает адаптированные версии операторов add, sub, div, mul и других для поддержки арифметических операций, но как явления отсутствуют привычные операторы перехода jmp, jz, jnz и другие. Так как байт-кода разрабатывался специально для использования его в ГА, операторы перехода и прямая адресация сделали бы фактически невозможным генерацию рабочего программного кода при помощи ГА из-за извечного обращения к несуществующим либо запрещённым адресам в потомках. Для вызовов функций сохранён серьезно измененный call, а также присутствуют высокоуровневые операторы if, else, while, позволяющие избежать прямой адресации, но при этом позволяющие строить условные операторы и определять циклы. Определены булевые операции, операции сравнения и переброски данных. Определены специфичные операции для работы с массивами (sort, min\_element и т.д.)).

Байт-код не предназначен для ввода-вывода данных, однако всё равно определяет 3 оператора ввода-вывода: puts (выводит С-строку), print (вывод переменной) и scan (ввод переменной). Подразумевается, что эти операторы будут использовать исключительно при отладке кода, и при генерации программных кодов в ГА они не будут использоваться.

**2.3.4 Пример программного кода**

Для демонстрации того, что байт-код достаточно полон и на нём возможно написать решение задачи коммивояжёра, представлены решение задачи коммивояжёра путём полного перебора на байт-коде (листинг В.1) и на алгоритмическом языке С++ (листинг В.2) для сравнения. В листинге В.1 все аргументы операций пишутся через пробельные символы, при этом параметры числа-адреса пишутся со знаком амперсанда (‘&’) в начале. Однострочные комментарии объявляются после символа ‘;’.

**[Приложение А](#_СОДЕРЖАНИЕ" \o "СОДЕРЖАНИЕ)**

Таблица 1 – Список команд байт-кода

|  |  |  |
| --- | --- | --- |
| **Имя** | **Параметры** | **Описание** |
| **Операции выделения памяти** | | |
| alloc\_int | int\* адрес | МассивУказателей и МассивТипов увеличиваются на 1. В конец МассивУказателей дописывается значение переменной по адресу-параметру. В конец МассивТипов дописывается «int». |
| alloc\_int\_const | int значение | МассивУказателей и МассивТипов увеличиваются на 1. В конец МассивУказателей дописывается параметр-значение. В конец МассивТипов дописывается «int». |
| alloc\_arr | int длина массива, int значения… | Операция, которая имеет всякий раз динамическое количество аргументов. МассивУказателей и МассивТипов увеличиваются на 1. В конец МассивУказателей дописывается результат выделения динамической памяти под arr\_int. Все элементы arr\_int инициализируются значениями, идущими после указания длины массива. В конец МассивТипов дописывается «arr\_int». |
| alloc\_arr\_const | int длина массива | МассивУказателей и МассивТипов увеличиваются на 1. В конец МассивУказателей дописывается результат выделения динамической памяти под arr\_int. Все элементы arr\_int по умолчанию равны 0. В конец МассивТипов дописывается «arr\_int». |
| alloc\_arr\_p\_const | int\* адрес\_по\_которому\_хранится\_длина\_массива | МассивУказателей и МассивТипов увеличиваются на 1. В конец МассивУказателей дописывается результат выделения динамической памяти под arr\_int. Все элементы arr\_int по умолчанию равны 0. В конец МассивТипов дописывается «arr\_int». |
| alloc\_arr\_copy | arr\_int\* адрес\_массива | МассивУказателей и МассивТипов увеличиваются на 1. В конец МассивУказателей дописывается результат выделения динамической памяти под arr\_int, размер и содержимое которого идентично массиву по адресу. В конец МассивТипов дописывается «arr\_int». |
| **Операции обращения к глобальным переменным** (graph, size) | | |
| graph | int колонка, int строка, int\* адрес\_записи | Запись по адресу значения graph[колонка][строка] |
| graph\_p | int\* колонка, int\* строка, int\* адрес\_записи | Запись по адресу значения graph[\*колонка][\*строка] |
| size | int\* адрес\_записи | Запись по адресу значения size |
| **Операции переброски данных** | | |
| move | int\* куда, int\* откуда | В переменную по первому адресу записывается значение переменной по второму адресу |
| move\_arr | arr\_int \* куда, arr\_int \* откуда | Массив по первому адресу становится копией массива по второму адресу |
|  |  |  |
| set | arr\_int\* адрес\_массива, int индекс, int\* адрес\_переменной | В элемент массива по индексу записывается значение переменной |
| get | arr\_int\* адрес\_массива, int индекс, int\* адрес\_переменной, | Значения элемента массива по индексу записывается в переменную |
| set\_p | arr\_int\* адрес\_массива, int\* адрес\_переменной\_с\_индексом, int\* адрес\_переменной | В элемент массива по индексу, хранящемуся по первому адресу, записывается значение переменной по второму адресу |
| get\_p | arr\_int\* адрес\_массива, int\* адрес\_переменной\_с\_индексом, int\* адрес\_переменной, | Значения элемента массива по индексу, хранящемуся по первому адресу записывается в переменную по второму адресу |
| move\_const | int\* куда, int константа | В переменную по адресу записывается параметр-значение |
| set\_const | arr\_int\* адрес\_массива, int индекс, int значение | В элемент массива по индексу записывается параметр-значение |
| set\_p\_const | arr\_int\* адрес\_массива, int адрес\_переменной\_с\_индексом, int значение | В элемент массива по индексу, хранящемуся по адресу, записывается параметр-значение |
| arr\_size | arr\_int\* адрес\_массива, int\* адрес\_записи | Сохранение по адресу записи длины массива |
| **Арифметические операции** | | |
| add | int\* адрес1, int\* адрес2, int\* результат | Сумма переменных по двум адресам записывается в адрес результата |
| sub | int\* адрес1, int\* адрес2, int\* результат | Разность переменных по двум адресам записывается в адрес результата |
| mul | int\* адрес1, int\* адрес2, int\* результат | Результат умножения переменных по двум адресам записывается в адрес результата |
| div | int\* адрес1, int\* адрес2, int\* результат | Результат деления первой переменной на вторую записывается в адрес результата |
| remainder | int\* адрес1, int\* адрес2, int\* результат | Остаток от деления первой переменной на вторую записывается в адрес результата |
| add\_const | int\* адрес1, int значение, int\* результат | Сумма переменной и параметра-значения записывается в адрес результата |
| sub\_const | int\* адрес1, int значение, int\* результат | Разность переменной и параметра-значения записывается в адрес результата |
| mul\_const | int\* адрес1, int значение, int\* результат | Результат деления переменной на параметр-значение записывается в адрес результата |
| div\_const | int\* адрес1, int значение, int\* результат | Остаток от деления переменной на параметр-значение записывается в адрес результата |
| remainder\_const | int\* адрес1, int значение, int\* результат |  |
| inc | int\* адрес | Инкремент переменной |
| dec | int\* адрес | Декремент переменной |
| **Операции сравнения и булевые операции** | | |
| not | int\* адрес, int\* результат | В результат записывается булевое не для переменной |
| or | int\* адрес1, int\* адрес2, int\* результат | В результат записывается булевое или для двух переменных |
| and | int\* адрес1, int\* адрес2, int\* результат | В результат записывается булевое и для двух переменных |
| equal | int\* адрес1, int\* адрес2, int\* результат | Если переменные по адресу равны, то запись в переменную-результат 1, иначе - 0 |
| more | int\* адрес1, int\* адрес2, int\* результат | Если переменная1 больше переменной2, то запись в переменную-результат 1, иначе - 0 |
| less | int\* адрес1, int\* адрес2, int\* результат | Если переменная1 меньше переменной2, то запись в переменную-результат 1, иначе – 0 |
| or\_const | int\* адрес1, int значение, int\* результат | В результат записывается булевое или переменной и параметра-значения |
| and\_const | int\* адрес1, int значение, int\* результат | В результат записывается булевое и переменной и параметра-значения |
| equal\_const | int\* адрес1, int значение, int\* результат | Если переменная по адресу и параметр-значение равны, то запись в переменную-результат 1, иначе - 0 |
| more\_const | int\* адрес1, int значение, int\* результат | Если переменная1 больше параметра-значения, то запись в переменную-результат 1, иначе - 0 |
| less\_const | int\* адрес1, int значение, int\* результат | Если переменная1 меньше параметра-значения, то запись в переменную-результат 1, иначе - 0 |
| **Операции сравнения, циклов, создания модуля и т.д.** | | |
| if, else, end\_if | int\* переменная | Если переменная не 0, то переход внутрь тела if, иначе – на следующую инструкцию после end\_if (либо после else, если присутствует). При поиске перехода читывается вложенность операторов |
| while, end\_while | int\* переменная | Пока переменная не 0 – проход внутри тела while. При поиске перехода читывается вложенность операторов |
| module, end\_module |  | \*Подсказка для алгоритма, что данный код работает прекрасно (понижает шансы на изменение кода, повышает шанс включения кода внутрь дочернего гена)\* |
| exit |  | Завершение программы |
| break |  | Производит принудительный выход цикла |
| continue |  | Принудительная новая итерация цикла |
| return |  | Выход из функции; вне функции эквивалента exit |
| function | int количество\_аргументов, int коды\_типов\_аргументов… | Операция, которая имеет всякий раз динамическое количество аргументов. При первой считке во время исполнения кода операция выделяет на стеке function\* с адресом функции. При последующих считываниях происходит исполнение функции. Аргументы: количество аргументов и список чисел: 0 – int, 1 – arr\_int, 2 – function, 3 – link\_int, 4 – link\_arr\_int, 5 – link\_function |
| end\_function |  | Метка конца функции |
| call | function\* адрес\_функции, <another type> аргументы… | Операция, которая имеет всякий раз динамическое количество аргументов. Совершает вызов функции с передачей указанных аргументов |
| **Дебаговый ввод-вывод** | | |
| puts | string строка | Только эта функция имеет в аргумент строку. Выводит ее на экран |
| print | <another type> адрес\_переменной | Операция выводит на экран значение переменной (переменная может быть адресом функции, адресом целого числа (выводится их значение) или массивом (выводится все содержимое массива)) |
| scan | int\* адрес\_переменной | По адресу записывается вводимое численное значение |
| **Операции для работы с массивами** | | |
| sort | arr\_int\* адрес\_массива, function\* компаратор | Вызов функции std::stable\_sort с данным массивом с функцией-компаратором вида function 3 3 3 3, где первые два аргумента – указатели на сравниваемые числа, последний - указатель на переменную, куда помещать результат |
| reverse | arr\_int\* адрес\_массива | Вызов функции std::reverse с данным массивом |
| min\_element | arr\_int\* адрес\_массива, function\* компаратор, int\* адрес\_результата | Вызов функции std::min\_element с данным массивом с функцией-компаратором вида function 3 3 3 3, где первые два аргумента – указатели на сравниваемые числа, последний - указатель на переменную, куда помещать результат. Результат (искомое число) сохраняется по адресу результата |
| max\_element | arr\_int\* адрес\_массива, function\* компаратор, int\* адрес\_результата | Вызов функции std::max\_element с данным массивом с функцией-компаратором вида function 3 3 3 3, где первые два аргумента – указатели на сравниваемые числа, последний - указатель на переменную, куда помещать результат. Результат (искомое число) сохраняется по адресу результата |
| next\_permutation | arr\_int\* адрес\_массива, int\* адрес\_результата | Вызов функции std::stable\_sort с данным массивом. Результат (флаг окончания поиска перестановок) сохраняется по адресу результата |

[**Приложение В**](#_СОДЕРЖАНИЕ)

#include <iostream>

#include <numeric>

#include <algorithm>

#include <iterator>

#include <vector>

using line = std::vector<uintmax\_t>;

using matrix = std::vector<line>;

constexpr auto INF = UINTMAX\_MAX;

std::ostream& operator<<(std::ostream& out, const line& obj) {

for (auto it = obj.cbegin(); it < obj.cend(); it++)

out << \*it + 1 << (it != obj.cend() - 1 ? ", " : "");

return out << std::flush;

}

int main(const int argc, const char\* const argv[]) {

const matrix matr = {

{INF, 1, 2, 3, 4},

{14, INF, 15, 16, 5},

{13, 20, INF, 17, 6},

{12, 19, 18, INF, 7},

{3, 2, 1, 0, INF}

};

size\_t good\_length = INF;

line result(matr.size(), 0);

std::generate(result.begin(), result.end(), []() {static uint64\_t i; return i++; });

line tmp = result;

do {

size\_t length = 0;

bool is\_not\_inf\_path = 0;

for (size\_t i = 0; i < tmp.size() - 1; i++) {

std::cout << tmp[i] << "\t" << tmp[i+1] << std::endl;

if (is\_not\_inf\_path = matr[tmp[i]][tmp[i + 1]] == INF)

break;

length += matr[tmp[i]][tmp[i + 1]];

}

length += matr[tmp[tmp.size() - 1]][tmp[0]];

if (is\_not\_inf\_path)

continue;

if (length < good\_length) {

good\_length = length;

result = tmp;

}

} while (std::next\_permutation(tmp.begin(), tmp.end()));

std::cout << "\n\n\n" << result << "\t\t" << good\_length << std::endl;

}

Листинг 1 - программа решения задачи коммивояжёра полным перебором на С++

; result - &0

alloc\_int\_const 9223372036854775807 ; good\_length - &1

alloc\_arr\_copy &0 ; tmp - &2

alloc\_int\_const 1 ; булевый флаг while - &3

while &3

alloc\_int\_const 0 ; length - &4

alloc\_int\_const 0 ; is\_not\_inf\_path - &5

alloc\_int\_const 0 ; i - &6

alloc\_int\_const 0 ; tmp.size() - 1 - &7

arr\_size &2 &7

dec &7

alloc\_int\_const 1 ; булевый флаг while - &8

while &8

alloc\_int &6 ; i - &9

alloc\_int &6 ; i + 1 - &10

inc &10

get\_p &2 &9 &9 ; теперь tmp[i] - &9

get\_p &2 &10 &10 ; теперь tmp[i+1] - &9

graph\_p &9 &10 &9 ; теперь matr[tmp[i]][tmp[i + 1]] - &9

equal\_const &9 9223372036854775807 &5

if &5

break

end\_if

add &4 &9 &4 ; length += matr[tmp[i]][tmp[i + 1]]

inc &6

less &6 &7 &8 ; i < tmp.size() - 1

end\_while

alloc\_int\_const 0 ; 0 - &9

alloc\_int &7 ; tmp.size() - 1 - &10

get\_p &2 &9 &9 ; tmp[0] - &9

get\_p &2 &10 &10 ; tmp[tmp.size() - 1] - &10

graph\_p &10 &9 &9 ; matr[tmp[tmp.size() - 1]][tmp[0]] - &9

add &4 &9 &4 ; length += matr[tmp[tmp.size() - 1]][tmp[0]]

if &5

break

end\_if

alloc\_int\_const 0 ; булевый флаг if - &11

less &4 &1 &11

if &11

move &1 &4

move\_arr &0 &2

end\_if

next\_permutation &2 &3

end\_while

print &0

puts "\t"

print &1

Листинг 2 - программа решения задачи коммивояжёра полным перебором на байт-коде