­Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и механики

**Высшая Школа Кибербезопасности и Защиты Информации**

**КУРСОВАЯ РАБОТА**

**Оптимизация курсовой работы**

по дисциплине «Структуры данных»

Выполнил

студент гр.3651001/80002 <*подпись*> Д. В. Михайлов

Руководитель,

Старший преподаватель доцент, <*подпись*> Е. В. Малышев

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 201\_\_ г.

Санкт-Петербург

2019

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| 1.1. Термины и определения. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| 1.2. Ход работы. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| Выводы. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |

**ВВЕДЕНИЕ**

Любую программу можно охарактеризовать некоторым занимаемым объемом памяти и определенным временем выполнения, поэтому чем она больше и сложнее, тем, соответственно, больше времени тратится на ее выполнение и тем большую память она будет занимать.

Для улучшения эффективности программы, необходимым является процесс оптимизации алгоритма, состоящий из четырех уровней: на уровне алгоритмов, на машинно-независимом уровне, на машинно-зависимом и на уровне компилятора. При всех модификациях необходимо помнить, что не всякое изменение может повлечь за собой только хороший результат. В качестве главного показателя оптимизации мною была выбрана скорость исполнения программы, а не по размеру, так как скорость работы является более важным фактором эффективности программы.

**1.1. ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ**

*Алгоритм* – набор инструкций, описывающий порядок действий исполнителя для достижения некоторого результата.

*Оптимизация* – модификация системы для повышения её эффективности. Система может быть одиночной компьютерной программой, цифровым устройством, набором компьютеров или даже целой сетью, такой как Интернет.

*Оптимизация алгоритмов* – модификация готовых алгоритмов для улучшения эффективности работы программы. За счет оптимизации невозможно добиться существенного улучшения алгоритма программы, можно только говорить об улучшении реализации этого алгоритма.

*Машинно–зависимая оптимизация* – оптимизация, которая зависит от выбора архитектуры компьютера.

*Машинно–независимая оптимизация* – оптимизация, которая направлена на снижение времени работы или объема программы. В этом случае преобразованию подвергается программа на уровне машинно-независимого промежуточного представления, общего для группы входных или машинных языков.

С точки зрения эффективности наиболее предпочтительной является *машинно-зависимая* оптимизация, поскольку именно с ее помощью можно учесть особенности конкретной̆ вычислительной̆ среды, однако машинно-зависимый оптимизатор непереносим. С другой стороны, преобразование программы на *уровне исходного языка* позволяет получить более эффективную программу, допускающую дальнейшее развитие и сопровождение. Наконец, *машинно-независимая* оптимизация на уровне промежуточного представления является компромиссом между этими двумя крайними случаями.

**1.2. Ход работы**

Для реализации методов оптимизации мною была взята курсовая работа из прошлого учебного семестра «Графическое представление АВЛ-деревьев», так как она содержит циклы, условия и выводов аргументов в консоль, то есть программа имеет потенциал к максимальному приросту эффективности с помощью оптимизации.

1. Машинно-независимая оптимизация.
   * Разъединение циклов

|  |  |
| --- | --- |
| ДО | ПОСЛЕ |
| while(node != nullptr)  {  if(node->value > value)  node = node->left;  else  {  if(node->value < value)  node = node->right;  else  {  node->marked = true;  break;  }  }  } | if(node->value > value)  {  while(node != nullptr)  node = node->left;  } else {  if(node->value < value)  {  while(node != nullptr)  node = node->right;  } else  while(node != nullptr)  {  node->marked = true;  break;  } } |

* Оптимизация переходов

|  |  |
| --- | --- |
| if(temp == p->left)  temp = p->right;  else  temp = p->left; | temp = (temp == p->left) ? p->right : p->left; |

* Правило де Моргана

|  |  |
| --- | --- |
| if(!first && !second)  return leftRotation(node); | if(!(first || second))  return leftRotation(node); |

* Устранение рекурсии

|  |  |
| --- | --- |
| Tree::amount(p->left);  Tree::amountValue += p->value;  Tree::amount(p->right); | while(current != nullptr) {  if(current->left == nullptr) {  Tree::amountValue += current->value;  current = current->right;  } else {  TreeNode^ pre = current->left;  while(pre->right!=nullptr&&pre->right!=current) {  pre = pre->right;  }  if(pre->right == nullptr) {  pre->right = current;  Tree::amountValue += current->value;  current = current->left;  } else {  pre->right = nullptr;  current = current->right;  }  }  } return amountValue; |

|  |  |  |
| --- | --- | --- |
| Количество элементов дерева | Время работы без рекурсии | Время работы с рекурсией |
| 10000 | 0.8595 ms | 0.6695 ms |
| 1000 | 0.5597 ms | 0.4714 ms |
| 100 | 0.4993 ms | 0.4530 ms |
| 50 | 0.4495 ms | 0.4369 ms |
| 10 | 0.4179 ms | 0.306 ms |

Как показывает практика, данная оптимизация не дает cильного прироста производительности, скорее даже может замедлить программу.

1. Машинно-зависимая оптимизация.

Использованы такие ключевые слова для переменных, как static, const, для функции - inline. Использование register оказалось бессмысленным, т.к. время выполнения не изменилось. Заменены типы всех переменных с int на short, т.к. согласно здравому смыслу, переменные в программе не будут принимать значения больше или меньше значений типа short.

|  |  |  |
| --- | --- | --- |
| Параметр | Флаг | Время работы |
| Оптимизация | Максимальная оптимизация (приоритет скорости) (/O2) | 7.754 ms |
| Предпочитать размер или скорость | Предпочитать скорость кода (/Ot) | 7.811 ms |
| Оптимизация всей программы | Да (/GL) | 7.659 ms |
| Опт. + Опт. всей прогр. | Макс. оптимизация (приоритет скорости) (/O2) + Да (/GL) | 6.460 ms |

1. Оптимизация компилятором.
2. Дополнительное задание.

Преобразование АВЛ-Дерева в Красно-черное дерево.

**АВЛ-дерево** — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

То есть - идеально сбалансированное дерево.

**Красно-черное дерево** — еще один вид самобалансирующихся деревьев, только дерево не идеально, а "примерно" сбалансированно. В каждую вершину добавляется еще одно поле - бит, отвечающий цвету (красный или черный). С помощью этого поддерживаются следующие свойства:

* Узел либо красный, либо чёрный.
* Корень — чёрный.
* Все листья — черные.
* Оба потомка каждого красного узла — черные.
* Всякий простой путь от данного узла до любого листового узла, являющегося его потомком, содержит одинаковое число черных узлов.

Эти ограничения реализуют критическое свойство красно-черных деревьев: путь от корня до самого дальнего листа не более чем в два раза длиннее пути от корня до ближайшего листа.

**Разница.**

Так как в идеально сбалансированном дереве много ресурсов тратится на поддержание сбалансированности, рекомендуется использовать его в ситуации, когда вставка/удаление происходит существенно реже считывания.

Красно-черные деревья, наоборот, тратят меньше ресурсов на поддержание сбалансированности (хотя здесь тоже есть перебалансировка), и их лучше использовать, когда вставка и чтение проиходят примерно с одинаковой частотой.

Таким образом, применив Красно-черное дерево вместо АВЛ, мы получим прирост производительности, следовательно и более оптимизированную программу.

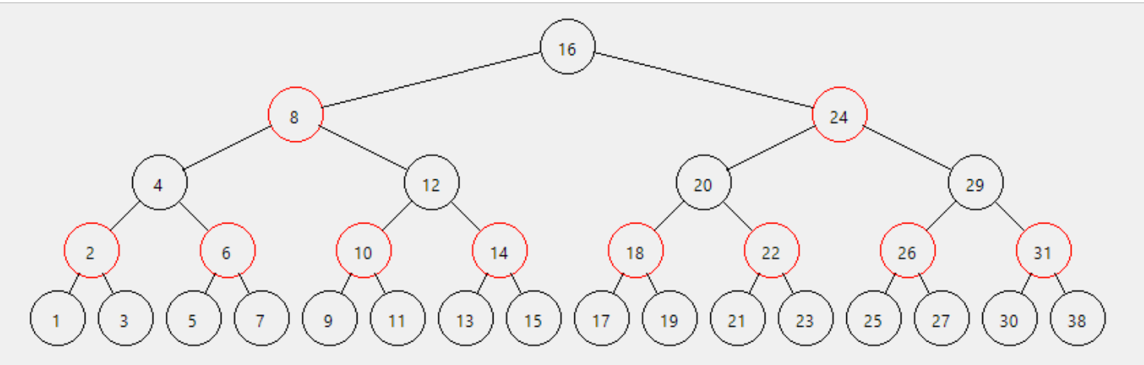
****

Рисунок 1. Результат преобразования дерева.

|  |
| --- |
| void Tree::AVLColor(TreeNode^ root)  {  if(root == nullptr)              return;         root->marked = false;         AVLColor(root->left);         AVLColor(root->right);         if(root->height % 2 == 1)         {             if(root->left != nullptr && root->left->height % 2 == 0)                 root->left->marked = true;             if(root->right != nullptr && root->right->height % 2 == 0)                 root->right->marked = true;         } |

Листинг 1. Algorithm conversion of AVL Tree to Red-Black.

**ВЫВОДЫ**

При выполнении данной лабораторной работы были получены комплексные знания по работе с разными видами оптимизации: в частности, оптимизации на уровне алгоритмов, машинно-зависимой и машинно-независимой оптимизациями, оптимизацией компилятора.

Как показала практика, наибольший прирост в скорости выполнения программы дала именно алгоритмическая оптимизация и распараллеливание циклов.

При выполнении дополнительного задания было реализовано преобразование АВЛ дерева в Красно-черное, которое дало прирост оптимизированности программы.