МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«**Вятский государственный университет**»

**(«ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №1

по дисциплине «Параллельное программирование»

Вариант 10

Выполнил студент группы ИВТ-32 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Щесняк Д. С./

Проверил доцент кафедры ЭВМ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Чистяков Г.А./

Киров 2016

1. Задание на лабораторную работу

Необходимо разработать алгоритм, позволяющий найти разрешающую последовательность ходов в пятнашках произвольной размерности.

1. Изучить алгоритм.
2. Провести доказательную оценку алгоритма по временной сложности и затратами по памяти.
3. Реализовать алгоритм с помощью языка С++.
4. Построить набор тестовых примеров (не менее 10) и провести оценку эффективности реализованного алгоритма
5. Изучение предметной области

Пятнашки — популярная головоломка, придуманная в 1878 году Ноем Чепмэном. Представляет собой набор одинаковых квадратных костяшек с нанесёнными числами, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры — перемещая костяшки по коробке, добиться упорядочивания их по номерам, желательно сделав как можно меньше перемещений.

Половина из всех возможных комбинаций пятнашек невозможно решить. Для определения «решаемости» комбинации применяется формула:

ni – количество квадратов меньше i-го считая от позиции i к 15.

e – номер строки, считая с 1, в котором находится пустая клетка.

Если N - нечетное число, то решения у данной комбинации нет.

1. Наивный алгоритм

Самый очевидный способ решения пятнашек заключается в проверке решаемости заданной комбинации и полный перебор всех возможных ходов.

Если размер поля 2x2, то количество вариантов расположения костяшек равно 24, из них могут быть решены только 12.

Если размер поля 3х3, то количество вариантов – 362880, из них возможно решить только 181440.

Как видно из приведенных выше примеров алгоритмическая сложность растет по факториалу O(n!).

Данная задача является NP-полной и не может быть решена за полиноминальное время. Существуют некоторые эвристики, позволяющие ускорить процесс вычисления, но на очень больших данных данная задача не сможет быстро выполниться.

При использовании данного алгоритма по факториалу возрастает не только сложность, но и затраты по памяти, так как необходимо хранить все состояния, во избежание зацикливания.

1. Алгоритм А\*

Данный алгоритм вносит в наивный алгоритм эвристику, благодаря которой значительно уменьшается число вариантов для перебора.

Последовательность шагов:

1. Вычислить функцию расстояния для начального положения.
2. Найти все соседние состояния от начального положения (все возможные сдвиги квадратов) и добавить в список не проверенных состояний
3. Добавить начальное состояние в список проверенных
4. Выбрать из списка непроверенных состояний то, которое имеет наименьшее расстояние.
5. Если расстояние равно 0, то перейти к пункту 9
6. Вычислить все соседние состояния
7. Добавить выбранное состояние в закрытый список
8. Перейти к пункту 4
9. Вывести результат

Количество итераций цикла напрямую зависит от выбранной функции расстояния.

В ходе практических экспериментов было выбрано Манхэттенское расстояние. Оно вычисляется данным образом: в цикле пробегаемся по строкам и столбцам матрицы и для каждого элемента вычисляем модуль разности между его текущим положением и положением, которое он должен занимать в собранной головоломке. Например, элемент 5 находится в 3-й строке и 3-м столбце (считая с 1), в отсортированной матрице он должен занимать позицию (2; 2), получается, что Манхэттенское расстояние равно 3 – 2 + 3 – 2 = 2. Данная функция расстояния позволяет за наименее короткий срок получить решение головоломки, но не обязательно наиболее короткое.

Для нахождения наиболее оптимального решения головоломки функция Манхэттенского расстояния немного модифицируется. К ее значение прибавляется количество передвижений квадратов, которые были совершенны в предыдущих состояниях.

Затраты памяти А\* так же растут со сложностью факториала, так как необходимо хранить все предыдущие состояния во избежание зацикливания алгоритма. Существует эвристика позволяющая хешировать значения игрового поля, хеши которых могут хранится в бинарных деревьях поиска, что ускоряет проверку наличия элементов в списках, но все так же необходимо хранить значения игрового поля, так как возможна ситуация коллизии.

За счет введения функции расстояния, алгоритмическая сложность алгоритма А\* всегда будет меньше наивного алгоритма, как и затраты памяти.

1. Программная реализация на С++

Модуль BinTree.h, реализующий двоичное дерево поиска для хранения состояний представлен на рисунке 1.

|  |
| --- |
| #pragma once  #include <cstdlib>  #include "Map.h"  #include "state.h"  struct Node {  Node\* left;  Node\* right;  State\* elem;  };  class BinTree  {  private:  Node\* node;  public:  int len;  BinTree(State\* s) {  this->node = new Node();  this->node->elem = s;  len = 1;  }  BinTree() {  this->node = new Node();  len = 0;  }  ~BinTree() {  }  int getLen() {  return len;  }  void add(State\* s) {  Node\* n = this->node;  len += 1;  if (n->elem == NULL) {  n->elem = s;  return;  }  do  {  if (s->getCost() >= n->elem->getCost()) {  if (n->right == NULL) {  n->right = new Node();  n->right->elem = s;  break;  }  else {  n = n->right;  }  }  else {  if (n->left == NULL) {  n->left = new Node();  n->left->elem = s;  break;  }  else {  n = n->left;  }  }  } while (true);  }  void del(State\* s) {  Node\* n = this->node;  Node\* p = NULL;  while (n->elem != s)  {  p = n;  if (s->getCost() > n->elem->getCost()) {  n = n->right;  }  else {  n = n->left;  }  if (n == NULL) return;  }  len -= 1;  if ((n->left == NULL) && (n->right == NULL)) {  if (p == NULL) {  n->elem = NULL;  return;  }  if (p->left == n) p->left = NULL;  else p->right = NULL;  delete n;  return;  }  if (n->left == NULL) {  if (p == NULL) {  this->node = n->right;  delete n;  return;  }  if (p->left == n) p->left = n->right;  else p->right = n->right;  delete n;  return;  }  if (n->right == NULL) {  if (p == NULL) {  this->node = n->left;  delete n;  return;  }  if (p->left == n) p->left = n->left;  else p->right = n->left;  delete n;  return;  }  if (n->right->left == NULL) {  n->elem = n->right->elem;  n->right = n->right->right;  return;  }  else {  Node\* k = n->right;  while (k->left->left != NULL) {  k = k->left;  }  n->elem = k->left->elem;  k->left = k->left->right;  }  }  //Компаратор при коллизии хешей  bool cmp(State\* a, State\* b) {  Map\* am = a->getMap();  Map\* bm = b->getMap();  int len = am->getCols() \* am->getLines();  for (int i = 0; i < len; i++) {  if (am->map[i] != bm->map[i]) return false;  }  return true;  }  //Поиск элемента в дереве  State\* find(State\* s) {  Node\* n = this->node;  if (n->elem == NULL) return NULL;  while ((n->elem->getHash() != s->getHash()) || (!cmp(n->elem, s)) ) {  if (s->getCost() >= n->elem->getCost()) {  n = n->right;  }  else {  n = n->left;  }  if (n == NULL) return NULL;  }  return n->elem;  }  //Поиск минимального элемента дерева  State\* min() {  Node\* n = this->node;  while (n->left != NULL) n = n->left;  return n->elem;  }  }; |

Рисунок 1 – Модуль BinTree.h

Исходный код модуля State.h, в котором описана структура класса состояния представлен на рисунке 2.

|  |
| --- |
| #pragma once  #include <cstdlib>  #include "map.h"  int costFunc(Map\* map);  unsigned int calcHash(Map\* map);  class State  {  public:  int cost;  State\* parent;  Map\* map;  int hash;  public:  State(Map\* map, State\* parent) {  this->cost = costFunc(map);  this->parent = parent;  this->map = map;  this->hash = calcHash(map);  }  State() {}  const State\* copy() const {  return this;  }  Map\* getMap() const {  return this->map;  }  int getHash() const {  return this->hash;  }  int getCost() const {  return this->cost;  }  State\* getParent() {  return this->parent;  }  };  int costFunc(Map\* map) {  int len = map->lines\*map->cols;  int sum = 0;  for (int i = 0; i < len; ++i)  {  if (map->map[i] == 0) continue;  int dx = abs((i % map->cols) - ((map->map[i] - 1) % map->cols));  int dy = abs((i / map->cols) - ((map->map[i] - 1) / map->cols));  sum += dx + dy;  }  return sum;  }  unsigned int calcHash(Map\* map) {  int len = map->lines \* map->cols;  unsigned int h = 0;  for (int i = 0; i < len; ++i) {  h += std::hash<int>{}(map->map[i] \* (1 << i));  }  return h;  } |

Рисунок 2 – Модуль State.h

Модуль Map.h, который описывает структуру игрового поля представлен на рисунке 3.

|  |
| --- |
| #pragma once  class Map  {  public:  int\* map;  int lines;  int cols;  public:  Map(int lines, int cols) {  this->map = new int[lines\*cols];  this->lines = lines;  this->cols = cols;  }  int getLines() {  return this->lines;  }  int getCols() {  return this->cols;  }  int find(int n) {  for (int i = 0; i < lines \* cols; ++i) {  if (map[i] == n) return i;  }  return -1;  }  Map\* shift(int angle) {  if ((angle < 0) || (angle > 3)) return this;  int len = lines \* cols;  Map\* mapc = new Map(lines, cols);  int ind = 0;  for (int i = 0; i < len; ++i) {  mapc->map[i] = map[i];  if (map[i] == 0) ind = i;  }  switch (angle) {  case 0: {  mapc->map[ind] = mapc->map[ind - cols];  mapc->map[ind - cols] = 0;  break;  }  case 1: {  mapc->map[ind] = mapc->map[ind + 1];  mapc->map[ind + 1] = 0;  break;  }  case 2: {  mapc->map[ind] = mapc->map[ind + cols];  mapc->map[ind + cols] = 0;  break;  }  case 3: {  mapc->map[ind] = mapc->map[ind - 1];  mapc->map[ind - 1] = 0;  break;  }  }  return mapc;  }  }; |

Рисунок 3 – Модуль Map.h

Исходный код файла main.cpp представлен на рисунке 4.

|  |
| --- |
| #include <iostream>  #include <cstdlib>  #include <cmath>  #include <ctime>  #include "binTree.h"  #include "Map.h"  #include "state.h"  #include <ctime>  #define TESTS 50  bool check(Map\*);  //Генератор игрового поля  Map\* generateMap(int lines, int cols) {  int len = lines\*cols;  Map\* map = new Map(lines, cols);  map->lines = lines;  map->cols = cols;  for (int i = 0; i < len; ++i)  {  map->map[i] = i + 1;  }  map->map[len - 1] = 0;  do  {  for (int i = 0; i < len - 1; ++i) {  int k = rand() % (len - 1);  int m = map->map[k];  map->map[k] = map->map[i];  map->map[i] = m;  }  } while (!check(map));  return map;  }  // Функция для вывода карты на экран  void printMap(Map\* map) {  std::cout << std::endl;  for (int i = 0; i < map->lines; ++i) {  for (int j = 0; j < map->cols; ++j) {  std::cout << map->map[i\*map->cols + j] << '\t';  }  std::cout << std::endl;  }  std::cout << std::endl;  }  //Функция проверки решаемости головоломки  bool check(Map\* map) {  int len = map->lines\*map->cols;  int n = 0;  int pos = 0;  for (int i = 0; i < len; ++i) {  if (map->map[i] == 0) {  pos = (i / map->cols) + 1;  continue;  }  for (int j = i; j < len; ++j) {  if (map->map[i] > map->map[j]) n += 1;  }  }  if (len & 1) {  if (n % 2 == 0) return true;  }  else {  if ((pos % 2 == 0) && (n & 1)) return true;  if ((pos & 1) && (n % 2 == 0)) return true;  }  return false;  }  //Функция поиска решения головоломки  std::vector<State\*> a(Map\* map) {  //Создаем открытый и закрытый список для состояний  BinTree open = BinTree(new State(map, NULL));  BinTree close = BinTree();  State\* min = open.min();  close.add(min);  open.del(min);  //Пока нет состояния в котором расстояние равно 0 (признак упорядоченности)  for(;min->getCost() != 0; min = open.min(), close.add(min), open.del(min))  {  //Находим пустую клетку  int zero = min->getMap()->find(0);  //Если свехру есть квадрат  if (zero / map->getCols() != 0) {  State\* s = new State(min->getMap()->shift(0), min);  if ((open.find(s) == NULL) && (close.find(s) == NULL)) {  open.add(s);  }  }  //Если справа есть квадрат  if (zero % map->getCols() != map->getCols() - 1) {  State\* s = new State(min->getMap()->shift(1), min);  if ((open.find(s) == NULL) && (close.find(s) == NULL)) {  open.add(s);  }  }  //Если снизу есть квадрат  if (zero / map->getCols() != map->getLines() - 1) {  State\* s = new State(min->getMap()->shift(2), min);  if ((open.find(s) == NULL) && (close.find(s) == NULL)) {  open.add(s);  }  }  //Если слева есть квадрат  if (zero % map->getCols() != 0) {  State\* s = new State(min->getMap()->shift(3), min);  if ((close.find(s) == NULL) && (open.find(s) == NULL)) {  open.add(s);  }  }  }  std::vector<State\*> solution;    State\* s = min;  do  {  solution.push\_back(s);  s = s->getParent();  } while (s != NULL);  return solution;  }  int main(int argc, char const \*argv[]) {  int lines, cols;  Map\* map;  std::cout << "Write matrix's dimensions: ";  std::cin >> lines >> cols;  std::cout << lines << " " << cols << std::endl;  if ( (lines <= 1) || (cols <= 1) ) {  std::cout << "wrong martix's parameters" << std::endl;  system("Pause");  exit(0);  }  double t = 0;  std::vector<State\*> ans;  for (int i = 0; i < TESTS; i++) {  srand(i);  map = generateMap(lines, cols);  std::cout << "Case #" << i + 1 << ": ";  clock\_t time = clock();  ans = a(map);  time = clock() - time;  std::cout << (double) time / 1000 << std::endl;  t += (double)time / 1000;  }  std::cout << "Average time = " << t / TESTS << std::endl;  system("pause");  return 0;  } |

Рисунок 4 – Исходный код main.cpp

1. Тестирование

Матрицы исходных головоломок представлены в таблицах 1-10. В таблице 1 описывается решенная головоломка размером 2х2. В таблице 2 описывается головоломка 2х2, которая не имеет решения. В таблице 3 описывается головоломка, которая имеет решение. В таблице 4 описывается головоломка 2х4. В таблицах 5-6 описываются матрицы головоломок 3х3. В таблицах 7,8 головоломки 4х4, в таблицах 9, 10 головоломки 5х5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 1   |  |  | | --- | --- | | 1 | 2 | | 3 | 0 | | Таблица 2   |  |  | | --- | --- | | 1 | 3 | | 2 | 0 | |
| Таблица 3   |  |  | | --- | --- | | 2 | 3 | | 1 | 0 | | Таблица 4   |  |  |  |  | | --- | --- | --- | --- | | 1 | 7 | 3 | 2 | | 5 | 6 | 4 | 0 | |
| Таблица 5   |  |  |  | | --- | --- | --- | | 3 | 7 | 1 | | 5 | 2 | 4 | | 8 | 6 | 0 | | Таблица 6   |  |  |  | | --- | --- | --- | | 2 | 6 | 1 | | 4 | 5 | 7 | | 3 | 8 | 0 | |
| Таблица 7   |  |  |  |  | | --- | --- | --- | --- | | 11 | 7 | 10 | 4 | | 8 | 5 | 3 | 13 | | 15 | 6 | 14 | 12 | | 9 | 2 | 1 | 0 | | Таблица 8   |  |  |  |  | | --- | --- | --- | --- | | 9 | 12 | 2 | 6 | | 13 | 10 | 4 | 5 | | 3 | 11 | 1 | 8 | | 15 | 7 | 14 | 0 | |
| Таблица 9   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 12 | 16 | 23 | 14 | 24 | | 17 | 6 | 3 | 11 | 2 | | 5 | 21 | 1 | 18 | 7 | | 13 | 20 | 8 | 9 | 19 | | 15 | 10 | 4 | 22 | 0 | | Таблица 10   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 8 | 23 | 22 | 14 | 20 | | 15 | 6 | 4 | 24 | 21 | | 9 | 12 | 16 | 3 | 2 | | 17 | 19 | 18 | 11 | 5 | | 7 | 1 | 10 | 13 | 0 | |
|  |  |

Решение головоломки таблицы 3 представлено на рисунке 5. Прохождение тестов программой представлено на рисунке 6.

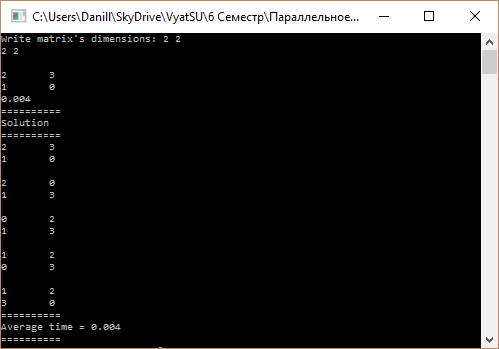


Рисунок 5 – результат решения головоломки 2х2

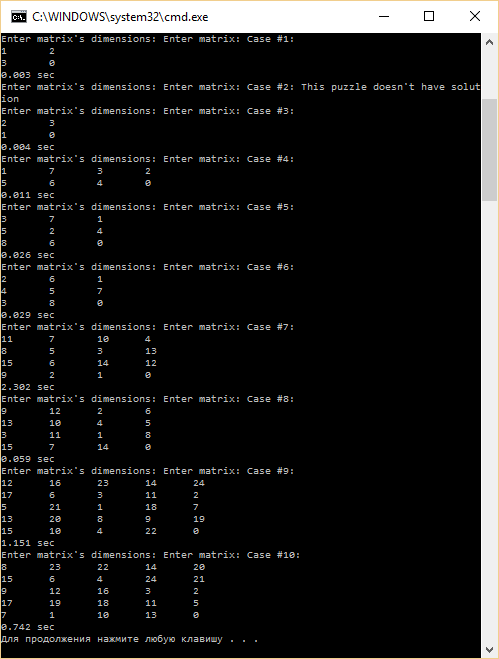


Рисунок 6 – результаты тестов

1. Вывод

В ходе выполнения лабораторной работы был разработан эффективный алгоритм решения игры пятнашки. Были рассмотрены наивный алгоритм и алгоритм A\* и были применены некоторые эвристики: использование бинарного дерева, хеширование игрового поля. В результате был разработан алгоритм, который за приемлимое время может найти решение игры для поля 5х5, а далее эта задача превращается в NP полную, из-за чего ожидание нахождения решения может затянуться на годы.