Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Чувашский государственный университет им. И.Н.Ульянова».
Кафедра вычислительной техники.
Предмет: Объектно-ориентированное программирование

Лабораторная работа №3.
Исчерпывающий поиск.

Вариант 5.

Выполнил: Васильев Егор Юрьевич
студент группы ИВТ-41-20

Чебоксары, 2021.

Цель работы – ознакомление с методами решения комбинаторных задач, получение навыков программирования оптимизационных задач.

Подготовка к работе.

1. Разработать алгоритм и программу решения задачи коммивояжера с использованием общей схемы решения методом ветвей и границ.

```
void traveling (int i, int n) {
    int j;
    if (i == (n + 1)) {
        if ((G[x[n - 1]][x[n]] != NoEdge) && ((cc + G[x[n]][1]) < bestc) && G[x[n]][1]
!= NoEdge) {
            for (j = 1; j < n; j++) {
                ans[j] = x[j];
            }
            bestc = cc + G[x[n]][1];
        }
    }
    else {
        for (j = i; j <= n; j++) {
            if ((G[x[i - 1]][x[j]] != NoEdge) && ((cc + G[x[i - 1]][x[j]]) < bestc)) {
                swap(x[i], x[j]);
                cc += G[x[i - 1]][x[i]];
                traveling(i + 1, n);
                cc -= G[x[i - 1]][x[i]];
                swap(x[i], x[j]);
            }
        }
    }
}
```

2. Разработать алгоритм и программу приближенного решения задачи коммивояжера

```
s=0
k=1
sum=0
function length_way(x){
for t=x to n do{
while k≤n−1 do {
            i:=t
            s+ct,i
            for j=1 to k do {
ci,aj= ∞
            }
       t=i
       k=k+1
       ak=t
}
return s
}
}
function min{
    Ввод t
    min=c[t,0]
    for j=0 to i do {
        if ct,i<min  then
       min=ct,i
    }
return min
}
```

```
x=0
mas
while x !=n-1 do {
   w=lenght_wayx
     insertmas,w
}
```

Индивидуальное задание.

Поиск с возвратом

Определить все возможные маршруты коня, начинающиеся на одном заданном поле шахматной доски и оканчивающиеся на другом. Никакое поле не должно встречаться в одном маршруте дважды

```
#include <iostream>
#include <iomanip>
#include <queue>
#include <climits>
#include <set>
#include <windows.h>

using namespace std;

struct Node {
    int x, y, dist_from_target;
    int x_prev, y_prev;

    Node (int x, int y, int dist_from_target = 0, int x_prev = -1, int y_prev = -1) {
        this->x = x;
        this->y = y;
        this->dist_from_target = dist_from_target;
        this->x_prev = x_prev;
        this->y_prev = y_prev;
    }

    bool operator < (const Node& obj)  const {
        return dist_from_target < obj.dist_from_target || dist_from_target ==
obj.dist_from_target && (x < obj.x || (x == obj.x && y < obj.y));
    }

    bool operator == (const Node& obj) const {
        return x == obj.x && y == obj.y && dist_from_target == obj.dist_from_target;
    }
};

bool isValid (int x, int y, int N) {
    return (x >= 0 && x < N) && (y >= 0 && y < N);
}

Node findShortestDistnace (int N, Node start, Node end, set <Node> &visited) {
    int row[] = {2, 2, -2, -2, 1, 1, -1, -1};
    int col[] = {-1, 1, 1, -1, 2, -2, 2, -2};

    queue <Node> queue;

    queue.push(start);
    while (!queue.empty()) {
        Node node = queue.front();
        queue.pop();

        int x = node.x, y = node.y, dist = node.dist_from_target;

        if (x == end.x && y == end.y) {
```

```
            visited.insert(node);
            return node;
        }

        if (!visited.count(node)) {
            visited.insert(node);

            for (int i = 0; i < 8; i++) {
                int x1 = x + row[i];
                int y1 = y + col[i];
                if (isValid(x1, y1, N)) {
                    queue.push({x1, y1, dist + 1, x, y});
                }
            }
        }
    }

    Node error = {start.x, start.y, INT_MAX, end.x, end.y};
    return error;
}

int main() {
    int N = 8;
    Node start = {0, 0};
    Node end = {7, 7};

    set <Node> visited;
    Node result = findShortestDistnace(N, start, end, visited);

    if (result.dist_from_target == INT_MAX) {
        cout << "Path not found";
    } else {
        cout << "The minimum number of steps: " << result.dist_from_target << endl <<
endl;
        cout << "One of the ways: " << endl;

        int **board = new int* [N];
        for (int i = 0; i < N; i++) {
            board[i] = new int[N]{0};
        }
        while (result.dist_from_target >= 0) {
            set <Node>::iterator it = visited.find(result);

            if (it != visited.end()) {
                board[result.x][result.y] = result.dist_from_target + 1;
                result.dist_from_target--;
                result.x = it->x_prev;
                result.y = it->y_prev;
            }
        }
        HANDLE handle = GetStdHandle(STD_OUTPUT_HANDLE);
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i][j] == 1) {
                    SetConsoleTextAttribute(handle, 3);
                    cout << setw(2) << "X";
                    SetConsoleTextAttribute(handle, 15);
                } else if (board[i][j] > 0){
                    SetConsoleTextAttribute(handle, 3);
                    cout << setw(2) << board[i][j] - 1;
                    SetConsoleTextAttribute(handle, 15);
                } else {
                    cout << setw(2) << board[i][j];
                }
            }
```

```
            cout << endl;
        }
    }
    return 0;
}
```

Метод решета
Найти все простые несократимые дроби, заключенные между 0 и 1, знаменатели которых не превышают 9 (дробь задается двумя натуральными числами – числителем и знаменателем).

```
#include <iostream>
#include <iomanip>

using namespace std;

int getNum (int x, int y) {
    if (!y) return x;
    return getNum(y, x % y);
}

int main () {
    int count = 0;
    for (int znam = 2; znam != 10; znam++) {
        for (int chis = 1; chis != znam; chis++) {
            if (getNum(znam, chis) == 1) {
                cout << ++count << ": " << chis << "/" << znam << endl;
            }
        }
    }
    return 0;
}
```

Метод ветвей и границ
Имеется m различных предметов, известны вес каждого предмета и его стоимость. Определить, какие предметы надо положить в рюкзак, чтобы общий вес не превышал заданной границы, а общая стоимость была максимальной.

```
#include <iostream>
#include <stack>
#include <iomanip>

using namespace std;

struct Item {
    double cost;
    double weight;
};

void init (int* num, Item items[]) {
    cout << "\n" << setw(27) << "List of the items:\n";
    for (int i = 0; i < *num; i++) {
        cout << setw(2) << i + 1 << '.';
        items[i].cost = (double)rand() / (double)RAND_MAX * (1000 + 10);
        cout << " Cost: " << fixed << setprecision(2) << setw(6) << right <<
items[i].cost;
        items[i].weight = (double)rand() / (double)RAND_MAX * (20 + 5);
```

```
        cout << setw(12) << "weight: "  << fixed << setprecision(2) << setw(5) <<
right << items[i].weight;
        cout << endl;
    }
}

void print (const int num, int chosen[], double maxWeight, double maxCost) {
    if (maxCost > 0) {
        cout << "\nTotal weight of items with numbers: ";
        for (int i = 0; i < num; i++) {
            if (chosen[i] == 1) {
                cout << i + 1 << " ";
            }
        }
    cout << "is lesser than " << maxWeight;
    cout << "\nTotal cost of this items: " << fixed << setprecision(2) << maxCost;
    }
}

void replace (const int num, int allItems[], int chosen[]) {
    for (int i = 0; i < num; i++) {
        chosen[i] = allItems[i];
    }
}

void calc(int num, Item items[], int arr[], double *maxWeight, double *maxCost) {
    if (arr[num] == 0) {
        *maxWeight -= items[num].weight;
        *maxCost -= items[num].cost;
    } else {
        *maxWeight += items[num].weight;
        *maxCost += items[num].cost;
    }
}

int main() {
    int num;
    double maxCost, maxWeight = 0;
    double cost = 0, weight = 0;
    int allItems[num + 1], chosen[num];
    Item item[num];

    cout << "Enter number of items: ";
    cin >> num;
    cout << "\nEnter max sum of items: ";
    cin >> maxWeight;

    init(&num, item);
```

```
    stack <int> stack;

    for (int i = num - 1; i >= 0; i--) {
        allItems[i] = 0;
        stack.push(i);
    }

    while (!stack.empty()) {
        int number = stack.top();
        stack.pop();
        allItems[number] = !allItems[number];
        calc(number, item, allItems, &weight, &cost);

        if ((weight <= max_weight) && (cost > maxCost)) {
            maxCost = cost;
            replace(num, allItems, chosen);
        }

        for (int j = number - 1; j >= 0; j--) {
            stack.push(j);
        }
    }

    print(num, chosen, maxWeight, maxCost);
}
```

Вывод: ознакомился с методами решения комбинаторных задач, получил навыки программирования оптимизационных задач.