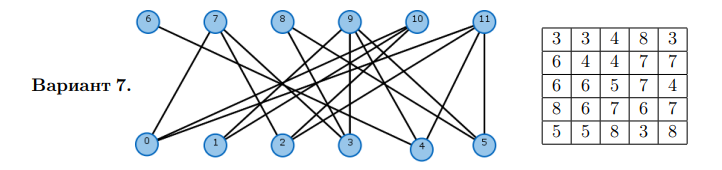
**Вариант 7**

Решите две задачи:

1. Найдите максимальное паросочетание и минимальное вершинное

покрытие в двудольном графе.

2. Решите задачу о назначениях.



**Решение задачи:**

**1**

Для решения данной задачи мы воспользуемся алгоритмом Куна нахождения наибольшего паросочетания в графе. Сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

Так же в двудольном графе размер минимального вершинного покрытия совпадает с размером максимального паросочетания.

**2**

Для решения задачи о назначениях воспользуемся венгерским алгоритмом. Алгоритм основан на двух идеях:

1. Если из всех элементов некой строки или столбца вычесть одно и то же число y, общая стоимость уменьшится на y, а оптимальное решение не изменится;
2. Если есть решение нулевой стоимости, то оно оптимально.

Алгоритм ищет значения, которые надо вычесть из всех элементов каждой строки и каждого столбца (разные для разных строк и столбцов), такие, что все элементы матрицы останутся неотрицательными, но появится нулевое решение.

**Решение С++:**

**1**

#include <iostream>

#include <vector>

using namespace std;

const int MAX\_V = 12;

vector<int> graph[MAX\_V];

vector<int> match(MAX\_V + 1, -1);

vector<bool> visited(MAX\_V + 1, false);

bool dfs(int u) {

if (visited[u])

return false;

visited[u] = true;

for (int v : graph[u]) {

if (match[v] == -1 || dfs(match[v])) {

match[v] = u;

return true;

}

}

return false;

}

int maximumMatching() {

int matchings = 0;

for (int u = 0; u < MAX\_V; ++u) {

fill(visited.begin(), visited.end(), false);

if (dfs(u))

++matchings;

}

return matchings;

}

void addEdge(int u, int v) {

graph[u].push\_back(v);

}

void printMatchingEdges() {

for (int v = 0; v < MAX\_V; ++v) {

if (match[v] != -1) {

cout << "{" << match[v] << ", " << v <<"}" << endl;

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

addEdge(0, 7);

addEdge(0, 10);

addEdge(0, 11);

addEdge(1, 9);

addEdge(1, 10);

addEdge(2, 7);

addEdge(2, 10);

addEdge(2, 11);

addEdge(3, 7);

addEdge(3, 8);

addEdge(3, 9);

addEdge(4, 6);

addEdge(4, 9);

addEdge(4, 11);

addEdge(5, 8);

addEdge(5, 9);

addEdge(5, 11);

int maxMatching = maximumMatching();

cout << "Максимальное паросочетание: " << maxMatching << endl;

int minVertexCover = MAX\_V - maxMatching;

cout << "Минимальное вершинное покрытие: " << minVertexCover << endl;

cout << "Ребра максимального паросочетания:" << endl;

printMatchingEdges();

return 0;

}

**2**

#include <iostream>

#include <vector>

#include <limits>

using namespace std;

const int MAX\_N = 5; // Размерность матрицы назначений

vector<vector<int>> assignments(MAX\_N, vector<int>(MAX\_N));

vector<int> matching(MAX\_N, -1);

vector<int> labelX(MAX\_N);

vector<int> labelY(MAX\_N);

vector<bool> visitedX(MAX\_N);

vector<bool> visitedY(MAX\_N);

bool dfs(int x) {

visitedX[x] = true;

for (int y = 0; y < MAX\_N; ++y) {

if (!visitedY[y]) {

int slack = labelX[x] + labelY[y] - assignments[x][y];

if (slack == 0) {

visitedY[y] = true;

if (matching[y] == -1 || dfs(matching[y])) {

matching[y] = x;

return true;

}

}

}

}

return false;

}

void updateLabels() {

int delta = numeric\_limits<int>::max();

for (int y = 0; y < MAX\_N; ++y) {

if (visitedY[y]) {

for (int x = 0; x < MAX\_N; ++x) {

if (!visitedX[x]) {

int slack = labelX[x] + labelY[y] - assignments[x][y];

delta = min(delta, slack);

}

}

}

}

for (int x = 0; x < MAX\_N; ++x) {

if (visitedX[x])

labelX[x] -= delta;

if (visitedY[x])

labelY[x] += delta;

}

}

int hungarianAlgorithm() {

for (int x = 0; x < MAX\_N; ++x) {

labelX[x] = numeric\_limits<int>::max();

labelY[x] = 0;

matching[x] = -1;

}

for (int y = 0; y < MAX\_N; ++y) {

for (int x = 0; x < MAX\_N; ++x) {

labelX[x] = min(labelX[x], assignments[x][y]);

}

}

for (int y = 0; y < MAX\_N; ++y) {

while (true) {

fill(visitedX.begin(), visitedX.end(), false);

fill(visitedY.begin(), visitedY.end(), false);

if (dfs(y))

break;

updateLabels();

}

}

int totalCost = 0;

for (int y = 0; y < MAX\_N; ++y) {

if (matching[y] != -1)

totalCost += assignments[matching[y]][y];

}

return totalCost;

}

int main() {

setlocale(LC\_ALL, "Russian");

assignments = {

{3, 3, 4, 8, 3},

{6, 4, 4, 7, 7},

{6, 6, 5, 7, 4},

{8, 6, 7, 6, 7},

{5, 5, 8, 3, 8}

};

int minimumCost = hungarianAlgorithm();

cout << "Минимальная стоимость назначений: " << minimumCost << endl;

return 0;

}

Ввод данных происходит в самой программе.

Вывод в консоль.

**Ответы:**

