|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САПР** | | Тема | оцінка | підпис |
| КН-414 | 4 | АЛГОРИТМ РІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА |  |  |
| Коцюба В.С. | |
| № залікової: 16081031 | |
| **Дискретні моделі в САПР** | | Викладач: | |
| к.т.н., асистент  Кривий Р.З. | |

**Мета:**

Метою даної лабораторної роботи є вивчення і дослідження алгоритмів рішення задачі комівояжера.

**Завдання:**

Написати програму для демонстрації роботи алгоритму задачі комівояжера.

**Теоретичні відомості:**

Умови існування гамільтонового контуру. Нижні границі.

Рішенням задачі комівояжера є оптимальний гамільтоновий контур. Нажаль, не всі графи містять гамільтоновий контур. Отже перед тим, ніж перейти до пошуку оптимального гамільтонового контура потрібно довести факт його існування в даному графі.

Можна знайти точний розв'язок задачі комівояжера, тобто, обчислити довжини всіх можливих маршрутів та обрати маршрут з найменшою довжиною. Однак, навіть для невеликої кількості міст в такий спосіб задача практично нерозв'язна. Для простого варіанта, симетричної задачі з n містами, існує (n − 1)! / 2 можливих маршрутів, тобто, для 15 міст існує 43 мільярдів маршрутів та для 18 міст вже 177 більйонів. Те, як стрімко зростає тривалість обчислень можна показати в наступному прикладі. Якщо існував би пристрій, що знаходив би розв'язок для 30 міст за годину, то для для двох додаткових міст в тисячу раз більше часу; тобто, більш ніж 40 діб.

Відомо багато різних методів рішення задачі комівояжера. Серед них можна виділити методи розроблені Белмором і Немхаузером, Гарфинкелем і Немхаузером, Хелдом і Карном, Стекханом. Всі ці методи відносяться до одного з двох класів: а) методи рішення, які завжди приводять до знаходження оптимального рішення, але потребують для цього, в найгіршому випадку, недопустимо великої кількості операцій(метод гілок та границь); б) методи, які не завжди приводять до находження оптимального результату, але потребують для цього допустимої великої кількості операцій (метод послідовного покращення рішення).

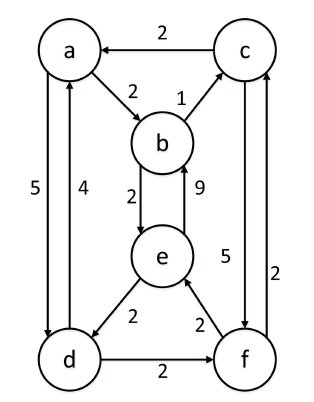


Рис.1 Заданий граф

**Робота з програмою:**

Після запуску програми в лівому краю вікна, у верхньому текст боксі задана початкова матриця суміжності графу, за потреби її там можна міняти.

Для запуску алгоритму натискаємо кнопку ‘Старт’, у нижньому текст боксі бачимо результати роботи алгоритму ( існує чи не існує маршрут), у центрі в полі канвас- відображення графа, а у правому текст боксі описані ребра, тобто вершини ребер та їх вага.

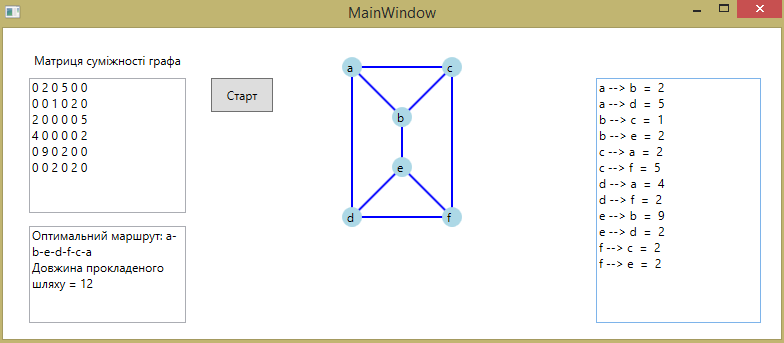


Рис.2 Вікно роботи програми

Фрагмент програми:

public String route(string alphabet)

{

String res = "";

for (int j = 1; j <= n; j++)

current\_way[j] = 0;

sum = 0;//занулюємо суму

end\_way = 0;//шлях вважаємо не знайденим

passedVertex = 1;

current\_way[1] = passedVertex; //починаємо обхід з першої вершини

minSum = Int32.MaxValue / 2;

recursSearch(1); //шукаємо починаючи з першої вершини

if (end\_way > 0) //якщо знайдено шлях

{

res += "Оптимальний маршрут: "; //починаємо формувати результуючу стрічку

int c = 1; //номер в порядку обходу вершин

for (int i = 1; i <= n; i++) //проходимо по всіх вершинах

{

int j = 1;

while ((j <= n) && (minWay[j] != c)) //шукаємо наступну вершину в порядку обходу

j++;

res += alphabet[j - 1] + "-"; //додаємо вершину до результуючої стрічки

c++;

}

res += alphabet[0];//до результуючої стрічки додаємо першу вершину, якою завершується обхід

res += "\nДовжина прокладеного шляху = " + minSum;//до результуючої стрічки додаємо суму ваг пройдених ребер

}

else

res = "Не вдалося знайти шлях.";

return res;

}

private void recursSearch(int x)

{

//якщо всі вершини переглянуті,

//і з останньої вершини є шлях в першу,

//і мнова сума відстаней менша мінімальної

if ((passedVertex == n) && (matrix\_for\_computations[x, 1] != 0) && (sum + matrix\_for\_computations[x, 1] < minSum))

{

end\_way = 1; //шлях вважається знайденим

minSum = sum + matrix\_for\_computations[x, 1]; //вводимо нову мінімальну суму відстаней

for (int i = 1; i <= n; i++)

minWay[i] = current\_way[i]; //вводимо новий мінімальний шлях

}

else

{

for (int i = 1; i <= n; i++) //переглядаємо всі вершини з поточної

//нова вершина не співпадає з біжучою, є прямий шлях з біжучої вершини в нову,

//нова вершина ще не переглянута, нова сума є меншою за мінімальну

if ((i != x) && (matrix\_for\_computations[x, i] != 0) && (current\_way[i] == 0) && (sum + matrix\_for\_computations[x, i] < minSum))

{

sum += matrix\_for\_computations[x, i]; //збільшуємо суму

passedVertex++; //збільшуємо кількість переглянутих вершин

current\_way[i] = passedVertex; //відмічаємо у нової вершини новий номер у порядку обходу

recursSearch(i); //пошук нової вершини починаючи з і (вершина, в яку перейшли)

current\_way[i] = 0; //повертаємо все назад passedVertex--;

sum -= matrix\_for\_computations[x, i];

}

}

}

Висновок: На цій лабораторній роботі було здійснено ознайомлення з алгоритмом рішення задачі комівояжера.