МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій

Кафедра систем штучного інтелекту

Лабораторна робота №5

з дисипліни
“Дискретна математика ”

Виконала:
ст. гр. КН-110
Кручковська Христина

Викладач:
Мельникова Н.І.

Львів – 2018

Тема:
Знаходження найкоротшого маршруту за алгоритмом Дейкстри. Плоскі планарні графи
Мета роботи:
набуття практичних вмінь та навичок з використання алгоритму Дейкстри.

ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ПРИКЛАДИ РОЗВ'ЯЗАННЯ ЗАДАЧ

Задача знаходження найкоротшого шляху з одним джерелом полягає у знаходженні найкоротших(мається на увазі найоптимальніших за вагою) шляхів від деякої вершини(джерела) до всіх вершин графа G . Для розв'язку цієї задачі використовується «жадібний» алгоритм, який називається алгоритмом Дейкстри. «Жадібними» називаються алгоритми, які на кожному кроці вибирають оптимальний із можливих варіантів. Задача про найкоротший ланцюг. Алгоритм Дейкстри. Дано n-вершинний граф G V, E , у якому виділено пару вершин v v V * 0 , , і кожне ребро зважене числом w(e) 0 . Нехай X x – множина усіх простих ланцюгів, що з'єднують 0 v з * v , Vx Ex x , . Цільова функція ( ) ( ) min Ex e F x w e . Потрібно знайти найкоротший ланцюг, тобто : x0 X ( ) min ( ) 0 F x F x xX Перед описом алгоритму Дейкстри подамо визначення термінів "k-а найближча вершина і "дерево найближчих вершин". Перше з цих понять визначається індуктивно так. 1-й крок індукції. Нехай зафіксовано вершину x0, E1 – множина усіх ребер e∈E, інцидентних v0. Серед ребер e∈E1 вибираємо ребро e(1) =(v0, v1), що має мінімальну вагу, тобто ((1)) min ( ) 1 w e w e eE . Тоді v1 називаємо першою найближчою вершиною (НВ), число w(e(1)) позначаємо l(1) = l(v1) і називаємо відстанню до цієї НВ.

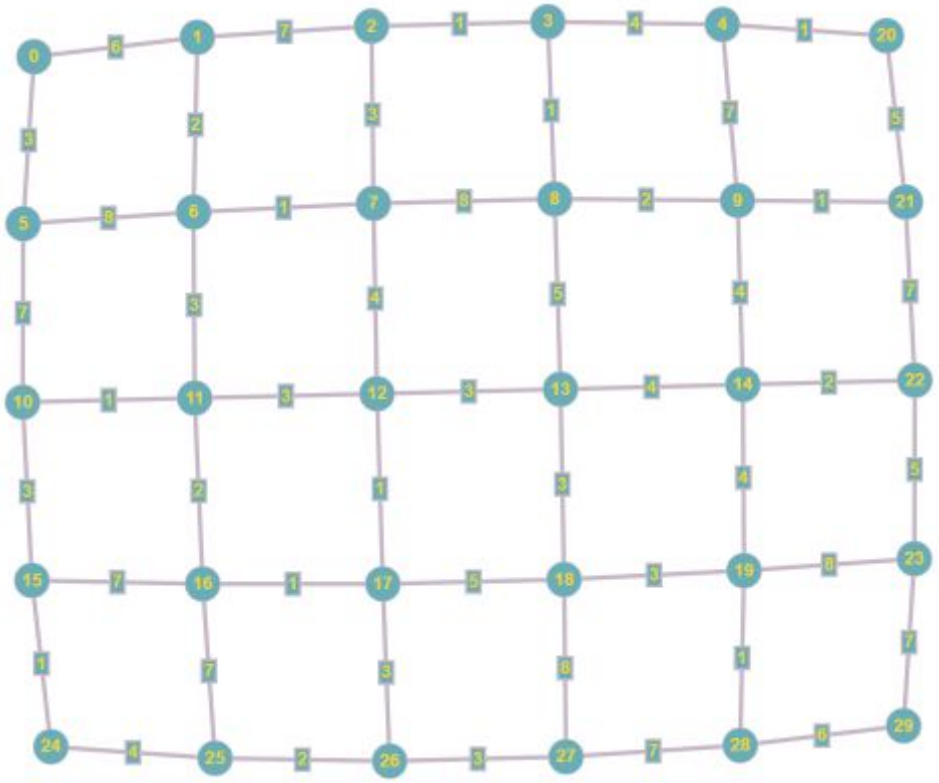
Позначимо V1={v0, v1} – множину найближчих вершин. 2-й крок індукції. Позначимо E2 – множину усіх ребер e=(v′,v″), e∈E, таких що v′∈V1, v″∈(V \V1). Найближчим вершинам v∈V1 приписано відстані l(v) до кореня v0 , причому l(v0)=0. Введемо позначення: V1 – множина таких вершин v″∈(V \V1), що ∃ ребра виду e =(v, v″), де v∈V1. Для всіх ребер e∈E2 знаходимо таке ребро e2=(v′, v2), що величина l(v′)+w(e2) найменша. Тоді v2 називається другою найближчою вершиною, а ребра e1, e2 утворюють зростаюче дерево для виділених найближчих вершин D2 ={e1, e2}. (s+1)-й крок індукції. Нехай у результаті s кроків виділено множину найближчих вершин Vs={v0, v1, …, vs} і відповідне їй зростаюче дерево Ds={e1, e2, …, es}... Для кожної вершини v∈Vs обчислена відстань l(v) від кореня v0 до v; Vs – множина вершин v∈(V\Vs), для яких існують ребра вигляду e =(vr , v), де vr ∈Vs , v∈(V\Vs). На кроці s+1 для кожної вершини vr∈Vs обчислюємо відстань до вершини vr : (1)( ) ( ) min (,) * * L s v l v w v v r v V r r s , де min береться по всіх ребрах e=(vr , v∗), v V s * , після чого знаходимо min серед величин L(s+1)(vr). Нехай цей min досягнуто для вершин vr0 і відповідної їй v V s * , що назвемо vs+1. Тоді вершину vs+1 називаємо (s+1)-ю НВ, одержуємо множину Vs+1 =Vs vs+1 і зростаюче дерево Ds+1 =Ds (vr0 ,vs+1). (s+1)-й крок завершується перевіркою: чи є чергова НВ vs+1 відзначеною вершиною, що повинна бути за умовою задачі зв'язано найкоротшим ланцюгом з вершиною v0 . Якщо так, то довжина шуканого ланцюга дорівнює l(vs+1)=l(vr0)+w(vr0, vs+1); при цьому шуканий ланцюг однозначно відновлюється з ребер зростаючого дерева Ds+1. У противному випадку випливає перехід до кроку s+2. Приклад. У графі на рис.5.1 знайти найкоротший ланцюг для виділеної пари вершин v0 , v∗. 9 8 7 6 v∗ v0 Розв’язання. Будемо позначати найближчі вершини v1, v2, v3, … у порядку їхньої появи (див. рис. 5.2): l(v1)=1, l(v2)=2, l(v3)=4, l(v4)=6, l(v5)=7, 30 60 24 5 50 71 40 91 1 1 1 3 5 9 2 4 34 6 17 18 14 15 1 1 4 3 2 Рисунок 5.1 Рисунок 5.2 l(v6)=9, l(v7)=11, l(v8)=16, l(v9)=18, l(v10)=19, l(v11)=19, l(v12)=20, l(v13)=20, l(v14)=22, l(v15)=40, l(v16)=41, l(v17)=49, l(v18)=56, l(v*)=62. Дерево найближчих вершин виділено на рисунку 5.2 жирними лініями і є кістяковим деревом, тому що містить усі

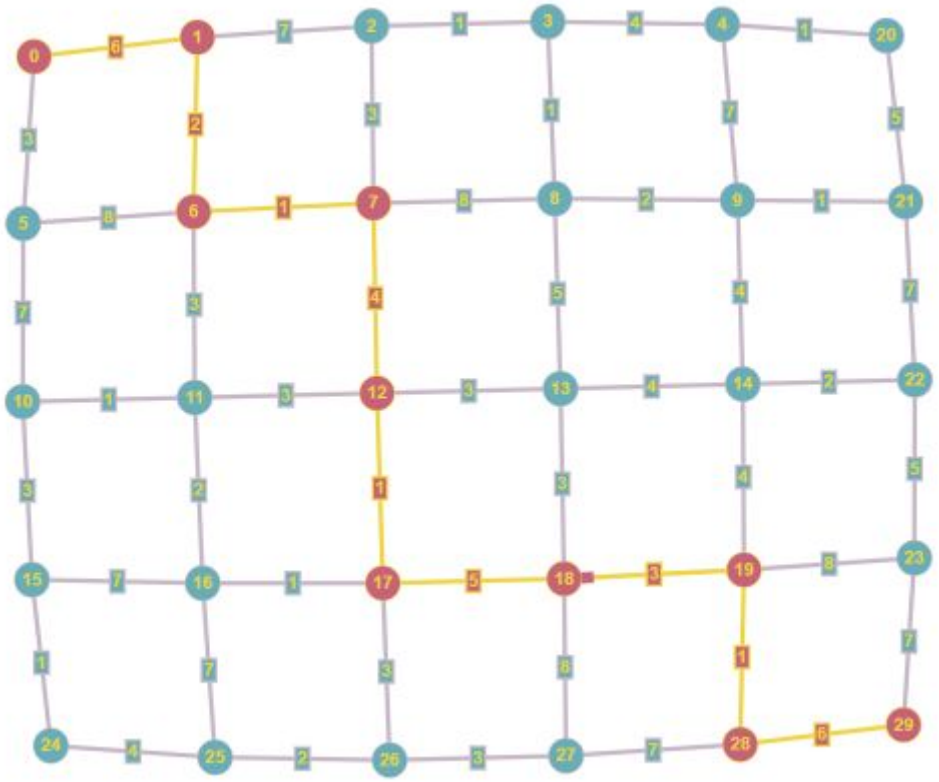
вершини графа. Шуканий найкоротший ланцюг: [v0, v1, v5, v7, v16, v17, v18, v ∗], довжина ланцюга l =l(v∗)=62. Плоскі і планарні графи Плоским графом називається граф, вершини якого є точками площини, а ребра – безперервними лініями без самоперетинань, що з'єднують відповідні вершини так, що ніякі два ребра не мають спільних точок крім інцидентної їм обом вершини. Граф називається планарним, якщо він є ізоморфним плоскому графу. Гранню плоского графа називається максимальна по включенню множина точок площини, кожна пара яких може бути з'єднана жордановою кривою, що не перетинає ребра графа. Границею грані будемо вважати множину вершин і ребер, що належать цій грані. V9 V4 V2 V1 V3 V6 V5 V10 V12 V11 V7 V8 V14 V13 V15 V16 V17 V18 V0 V * Алгоритм укладання графа G являє собою процес послідовного приєднання до деякого укладеного підграфа G ~ графа G нового ланцюга, обидва кінці якого належать G ~ . При цьому в якості початкового плоского графа G ~ вибирається будь-який простий цикл графа G . Процес продовжується доти, поки не буде побудовано плоский граф, ізоморфний графові G , або приєднання деякого ланцюга виявиться неможливим. В останньому випадку граф G не є планарным. Нехай побудоване деяке укладання підграфа G ~ графа G . Сегментом S відносно G ~ будемо називати підграф графа G одного з наступних виглядів: - ребро e E , e u, v , таке, що e E ~ , , ~ u,v V G V E ~ , ~ ~ ; - зв'язний компонент графа G – G ~ , доповнений всіма ребрами графа G , інцидентними вершинам узятого компонента, і кінцями цих ребер. ~ Вершину v сегмента S відносно G будемо називати контактною, якщо v V ~ . Припустимою гранню для сегмента S відносно G ~ називається грань Г графа G ~ , що містить усі контактні вершини сегмента S. Через Г(S) будемо позначати множину припустимих граней для S. Назвемо α-ланцюгом простий ланцюг L сегмента S, що містить дві різні контактні вершини і не містить інших контактних вершин. Тепер формально опишемо алгоритм . 0. Виберемо деякий простий цикл C графа G і укладемо його на площині; покладемо G G ~ . 1. Знайдемо грані графа G ~ і сегменти відносно G ~ . Якщо множина сегментів порожня, то перейдемо до пункту 7. 2. Для кожного

сегмента S визначимо множину Г(S). 3. Якщо існує сегмент S, для якого Г(S)=∅, то граф G не планарний. Кінець. Інакше перейдемо до п. 4. 4. Якщо існує сегмент S, для якого мається єдина припустима грань Г, то перейдемо до п. 6. Інакше до п. 5. 5. Для деякого сегмента S Г(S)>1. У цьому випадку вибираємо довільну припустиму грань Г. 6. Розмістимо довільний α- ланцюг L∈S у грань Г; замінимо G ~ на G L ~ і перейдемо до п. 1. 7. Побудовано укладання G ~ графа G на площині. Кінець. Кроком алгоритму будемо вважати приєднання до G ~ α- ланцюга L.

Завдання 1

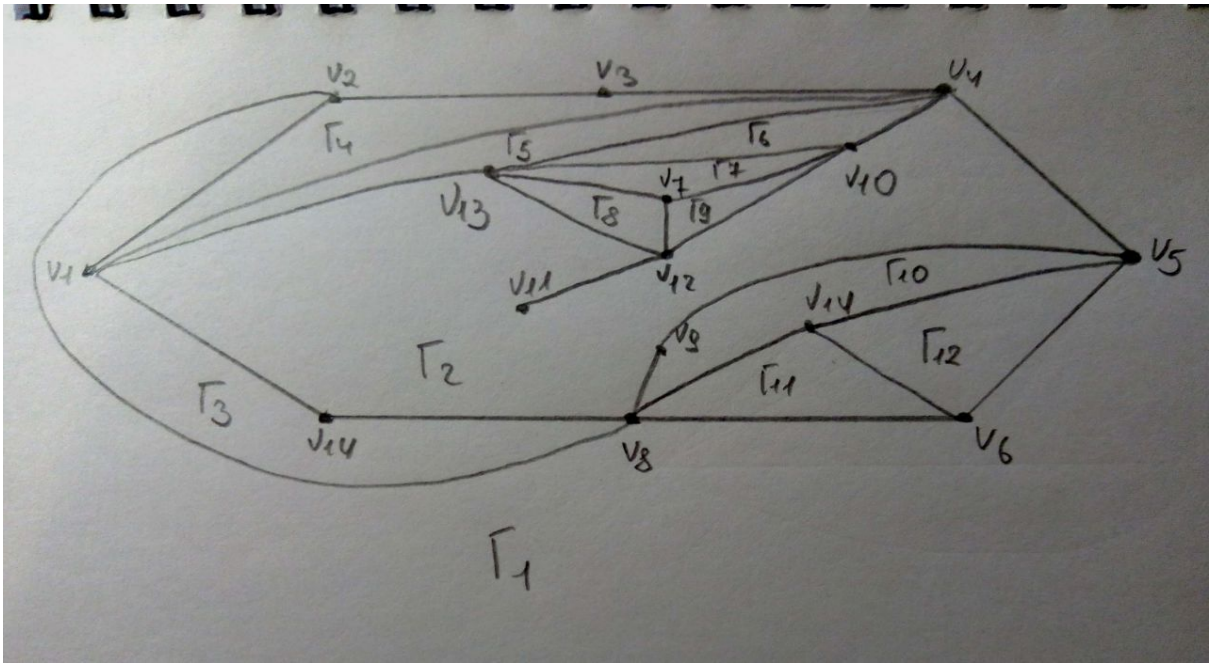
1. За допомогою алгоритму Дейкстра знайти найкоротший шлях у графі поміж парою вершин V0 і V * .





За допомогою γ-алгоритма зробити укладку графа у площині, або довести що вона неможлива.

Укладка цього графа неможлива, бо ребро V6-V14 буде перетинати інші в будь-якому випадку.



Завдання 2

Написати програму, яка реалізує алгоритм Дейкстри знаходження найкоротшого шляху між парою вершин у графі. Протестувати розроблену програму на графі згідно свого варіанту.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

short ver_am = 0;
struct vert *list[20];
short veco = 0;

struct vert{ short id;
    short degr;
    short used;
    short way;
    struct vert *prev;
    struct edge *edge[10];
};

struct edge{
    short v1id;
    short v2id;
    short wght;
    short used;
    struct vert *v1pt;
    struct vert *v2pt;
};


// ---Announcement of functions--- //
struct vert *create();                                  // Function for
creating vertices struct
```

```
void connection(struct vert *a, struct vert *b, short weight);  // Function for creating edges
void start(struct vert *vert);                                  // Function to choose start verticle
void dijkstra(struct vert *vert);
void way(struct vert *vert);                                    // Function that shows minimum way


// ---Main function--- //
int main(){
   // ---Create verticles--- //
   struct vert *v0 = create();    struct vert *v1 = create();    struct vert *v2 = create();
   struct vert *v3 = create();    struct vert *v4 = create();    struct vert *v5 = create();
   struct vert *v6 = create();    struct vert *v7 = create();    struct vert *v8 = create();
   struct vert *v9 = create();    struct vert *v10 = create();   struct vert *v11 = create();
   struct vert *v12 = create();   struct vert *v13 = create();   struct vert *v14 = create();
   struct vert *v15 = create();   struct vert *v16 = create();   struct vert *v17 = create();
   struct vert *v18 = create();   struct vert *v19 = create();   struct vert *v20 = create();
   struct vert *v21 = create();   struct vert *v22 = create();   struct vert *v23 = create();
   struct vert *v24 = create();   struct vert *v25 = create();   struct vert *v26 = create();
   struct vert *v27 = create();   struct vert *v28 = create();   struct vert *v29 = create();

   // ---Weight of edges---//
```

```
    connection(v0,v1,6);   connection(v0,v6,3);
connection(v1,v2,7);   connection(v1,v7,2);
    connection(v2,v3,1);   connection(v2,v8,3);
connection(v3,v4,4);   connection(v3,v9,1);
    connection(v4,v5,1);   connection(v4,v10,7);
connection(v5,v11,5);  connection(v6,v7,8);
    connection(v6,v12,7);  connection(v7,v8,1);
connection(v7,v13,3);  connection(v8,v9,8);
    connection(v8,v14,4);  connection(v9,v10,2);
connection(v9,v15,5);  connection(v10,v11,1);
    connection(v10,v16,4); connection(v11,v17,7);
connection(v12,v13,1); connection(v12,v18,3);
    connection(v13,v14,3); connection(v13,v19,2);
connection(v14,v15,3); connection(v14,v20,1);
    connection(v15,v16,4); connection(v15,v21,3);
connection(v16,v17,2);
    connection(v16,v22,4); connection(v17,v23,5);
connection(v18,v19,7); connection(v18,v24,1);
    connection(v19,v20,1); connection(v19,v25,7);
connection(v20,v21,5); connection(v20,v26,3);
    connection(v21,v22,3); connection(v21,v27,8);
connection(v22,v28,1); connection(v22,v23,8);
    connection(v23,v29,7); connection(v24,v25,4);
connection(v25,v26,2); connection(v26,v27,3);
    connection(v27,v28,7); connection(v28,v29,6);

    start(v0);
    printf("\n"); way(v29);
    printf("The shortest way is \x1b[32m%i\x1b[0m\n\n", v29->way);
}


struct vert *create(){
    struct vert *vert = (struct vert*)
    malloc(sizeof(struct vert));
```

```
    vert->id = ver_am; vert->degr = 0;
    vert->used = 0;
    vert->way = SHRT_MAX;
    ver_am++;
    return vert;
}

//function for creating edges
void connection(struct vert *a, struct vert *b, short weight){
    struct edge *edge = (struct edge*) malloc(sizeof(struct edge));
    edge->v1id = a->id; edge->v2id = b->id; edge->wght = weight;
    edge->v1pt = a; edge->v2pt = b;
    a->edge[a->degr] = edge; a->degr++;
    b->edge[b->degr] = edge; b->degr++;
}

void start(struct vert *vert){
    vert->way = 0; vert->prev = NULL; dijkstra(vert) ;
}

void dijkstra(struct vert *vert){
    list[veco] = vert; vert->used = 1; veco++;
    for (short j = 0; j < veco; j++){
        for (short i = 0; i < list[j]->degr; i++){
            if (list[j]->way + list[j]->edge[i]->wght <
list[j]->edge[i]->v2pt->way){
                list[j]->edge[i]->v2pt->prev = list[j];
                list[j]->edge[i]->v2pt->way = list[j]->way +
list[j]->edge[i]->wght;
            }
            if (list[j]->way + list[j]->edge[i]->wght <
list[j]->edge[i]->v1pt->way){
                list[j]->edge[i]->v1pt->prev = list[j];
                list[j]->edge[i]->v1pt->way = list[j]->way +
list[j]->edge[i]->wght;
```

```
            }
        }
        for (short i = 0; i < list[j]->degr; i++){
            if (list[j]->edge[i]->v2pt->used == 0)
dijkstra(list[j]->edge[i]->v2pt); if (list[j]->edge[i]->v1pt->used == 0)
dijkstra(list[j]->edge[i]->v1pt);
        }
    }
}

void way(struct vert *vert){
    if (vert->prev != NULL){
        printf("v%d <-- ", vert->id);
        way(vert->prev);
    }
    else if (vert->prev == NULL)
        printf("v%d\n", vert->id);
}
```

Висновок: я набула практичних вмінь та навичок з використання алоритму Дейкстри, а також навчилась укладати граф.