УКРАЇНСЬКИЙ КАТОЛИЦЬКИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНИХ НАУК

Назва проекту:

Задача 2-SAT

Автори: Наумець Захар

Нагурна Ольга

Вострикова Альона

Намасний Андрій

21 грудня 2021

Вступ

Реалізація проекта

Для створення комп'ютерного проекту з дискретної математики ми обрали тему 8 - «Задача 2-SAT», вирішення якої полягає в тому , щоб у неорієнтованому графі змінити колір на новий, відмінний від попереднього фарбуванням та відмінний від кольору суміжних вершин. Писання нашого проекту ми вирішили здійснити на Python. В коді ми створили функцію read( ), що читає csv файл та і повертає нам інформацію про вершини і їх кольори у вигляді масивів. Також у нас є функція main( ) , що викликає у собі більшість головних функцій (dfs1( ), dfs2( ), dfs3( )). Дані функції відповідають за аналіз графу та використання основних його властивостей. Під час роботи над проектом, ми використовували отриманні знання про неорієнтовні графи та компоненти зв'язності в неорієнтованих графах. У функціях dfs( ) ми використовували поняття пошуку вглиб та «стек»(останній зайшов - останній вийшов) , щоб обійти всі вершини з точністю один раз та добавляти пройдені елементи в інший список вершин, кольори яких ми вже перевірили. Також ми застосували вивчені знання про розфарбування графів, що ніякі дві суміжні вершини не можуть бути зафарбовані різними кольорами.

```
def read(path):
    global graph_init, graph_init_colors, num
    transf = {'red' : 1, 'green' : 2, 'blue' : 3}
    num = 0
    colors = []
    edges = []
    with open(path, 'r') as file:
        f = csv.reader(file)
        for i, line in enumerate(f):
            if i == 0:
                continue
            edges.append([int(line[0]) - 1, int(line[1]) - 1])
            colors.append((int(line[0]) - 1, int(transf[line[2]])))
            colors.append((int(line[1]) - 1, int(transf[line[3]])))
            num = max(int(line[0]), int(line[1]), num)
    graph_init = [[] for i in range(num)]
    graph_init_colors = [[] for i in range(num)]
```

1)В функції read, яка аргументом приймає шлях до файлу, ми зчитуємо файл, який містить інформацію про граф.
Для зручності ми перетворюємо кольори в числа

graph_init - список, який складається з таплів(з якими вершинами з'єднана ця вершина)
graph_init_colors - список, який складається з таплів(які кольори може приймати ця вершина)
num - к-ть вершин

edges - список, який складається з вершин
colors - список, який складається з таплів (вершина, її колір)

```
graph_init[i[0]].append(i[1])
# на індекс першої вершини додаємо 2 вершину
graph_init[i[1]].append(i[0])
# на індекс другої вершини додаємо 1 вершину
```

Далі ми працюємо з кольорами

```
for i in colors:
        if not len(graph_init_colors[i[0]]):
            for j in range(1, 4):
                if j != i[1]:
                    graph_init_colors[i[0]].append(j)
```

ми перевіряємо чи список за індексом вершини в graph_init_colors пустий, і якщо так ми запускаємо цикл (1,4) і перевіряємо, якщо j!= кольору вершини, то на індекс тієї вершини додаємо це j, тобто всі можливі кольори, які може приймати ця вершина.
2) Потім ми знову повертаємось до main() та рухаємось далі

```
for i, _ in enumerate(graph_init):
        graph_colors[i] = graph_init_colors[i][0]
        graph_colors[i + num] = graph_init_colors[i][1]
```

Створюємо масив, в який записуємо дві варіації вершини з двома можливими кольорами

```
for i, edges in enumerate(graph_init):
        for j in edges:
            if graph_init_colors[i][0] == graph_init_colors[j][0]:
                graph[i].append(num + j)
            if graph_init_colors[i][0] == graph_init_colors[j][1]:
                graph[i].append(j)
            if graph_init_colors[i][1] == graph_init_colors[j][0]:
                graph[i + num].append(j + num)
            if graph_init_colors[i][1] == graph_init_colors[j][1]:
                graph[i + num].append(j)
```

Перевіряємо чи містять сусідні вершини однакові можливі кольори і в залежності від цього, записуємо в новий список.

```
for i in range(2 * num):
        if not grey[i]:
            stack.append(i)
            while len(stack):
                dfs1()
```

Перевіряємо чи i-тий елемент є grey(тобто ще не поміченим), якщо так,то до stack додаємо індекс i.
Поки stack містить елементи, запускаємо ф-ію dfs1()
3)

```
def dfs1():
if black[ver]:
       sequence.append(ver)
       stack.pop()
       return
   grey[ver] = 1
   if grey[ver]:
       black[ver] = 1
   for next in graph[ver]:
       if not grey[next]:
           grey[next] = 1
           stack.append(next)
```

У цій функції ми перевіряємо вершини(чи вони сірі чи чорні), тобто чи пройшли ми вже їх, чи вони досі в черзі, чи вони ще поза чергою.
Ми використовуємо stack для того, щоб працювати з вершинами.
4)І знову до main()

```
for i, edges in enumerate(graph):
       for j in edges:
           Tgraph[j].append(i)
```

проходимось по вершинах з якими наша вершина* і має спільні можливі кольори
в список Tgraph на індекс тих вершин додаю нашу найпершу вершину*

```
while len(sequence):
       ver = sequence[-1]
       sequence.pop()
       if not grey[ver]:
           stack.append(ver)
           while len(stack):
               dfs2()
           color += 1
```

Ми заходимо в чергу, беремо останній елемент(вершину).
Перевіряємо чи заходили ми в цю вершину або чи додавали її до черги
Створюємо зміну з кольором і запускаємо ф-ію dsf2()
5)

```
def dfs2():
# записуємо компоненту нашої вершини
```

```
comp[ver] = color
   grey[ver] = 1
   stack.pop()
   for next in Tgraph[ver]:
       if not grey[next]:
           grey[next] = 1
           stack.append(next)
```

ми проходимось по вершинках, яка кольорами зв'язана з нашою вершинкою і якщо її немає в грей(тобто вона не сіра), то робимо її сірою і добавляємо до stack

6)І знову до main()

```
for i in range(num):
       if comp[i] == comp[num + i]:
           print("it is impossible")
           return
```

Перевіряємо чи наша вершина не міститься в одній компоненті

```
res_col = [0 for i in range(num)]
```

7)Список з фінальними кольорами

```
def dfs3():
if black[ver]:
       stack.pop()
       ver1 = ver
       if ver >= num:
           ver -= num
       if not res_col[ver]:
           res_col[ver] = graph_colors[ver1]
       return
```

Ми нормалізуємо нашу вершинку(робимо її в range(1, num)) і якщо її ще немає в фінальному списку, то додати її туди

```
for next in graph[ver]:
       if not grey[next]:
           grey[next] = 1
           stack.append(next)
```

Якщо наступна вершина, ще не була в черзі до додаємо її до stack

Висновок

За період працювання ми навчилися застосовувати отримані знання не тільки на практиці , але й оптимізовувати їх за допомогою програмування, що значно пришвидшить всі необхідні процеси. Також ми отримали цінний досвід роботи в команді та навчилися працювати над помилками.