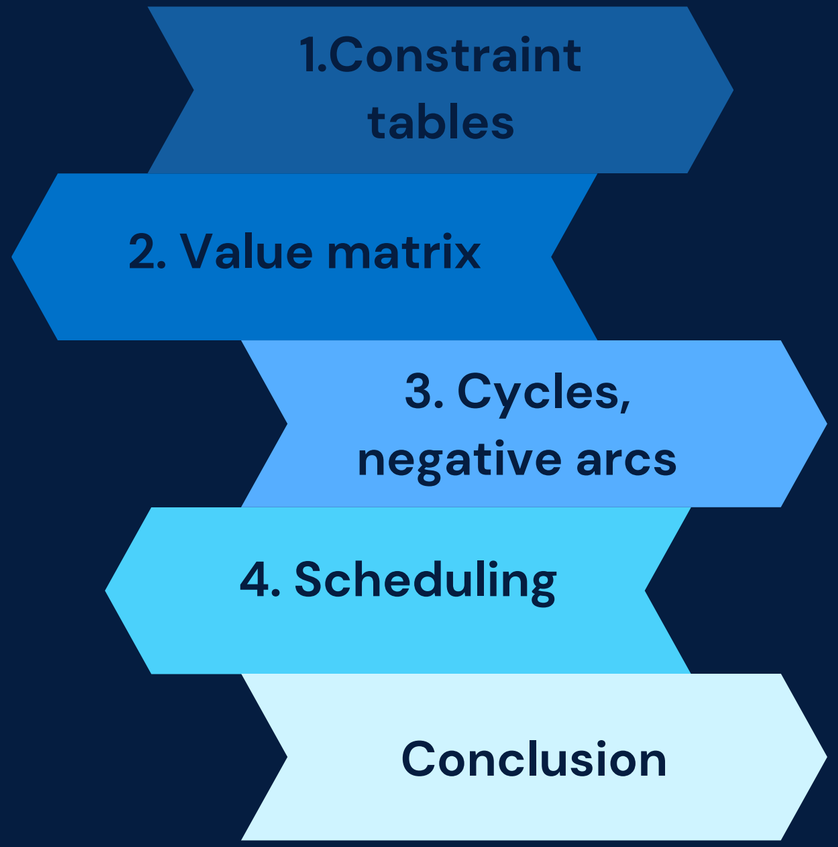
GRAPH THEORY

Presented by: Antoine Coustets
Naël Duval
Farès Soudani
Alejandro Charland

Overview


1.Constraint tables
2. Value matrix
3. Cycles, negative arcs
4. Scheduling
Conclusion

INTRODUCTION

Python

Versatility and readability, rich ecosystem and libraries, efficiency and scalability, universal code language

Github

Divise tasks, work remotely, share codes, look for the developpement of the project

1. CONSTRAINT TABLES

```python
def read_one_text(index):
    file = open('Constraint_Tables/table_' + str(index) + '.txt')
    data = []
    for row in file:
        data.append([str(x) for x in row.split()])

    return data
```

The read_one_text function reads a constraints table from a text file identified by an index. It returns the list of extracted constraints, each represented by a list of words.

```python
def transform_pretty_table(matrice):
    a = len(matrice[0])
    if a == 3 and matrice[0][0] != 'Task':
        matrice.insert(0, ['Task', 'Duration', 'Predecessors'])
    print(tabulate(matrice, headers='firstrow', tablefmt='fancy_grid'))
```

transform_pretty_table checks for a header row and adds one if missing (Task, Duration, Predecessors). Then it formats the matrix as a table using tabulate library.

Result:

Here the constraint table n° 8

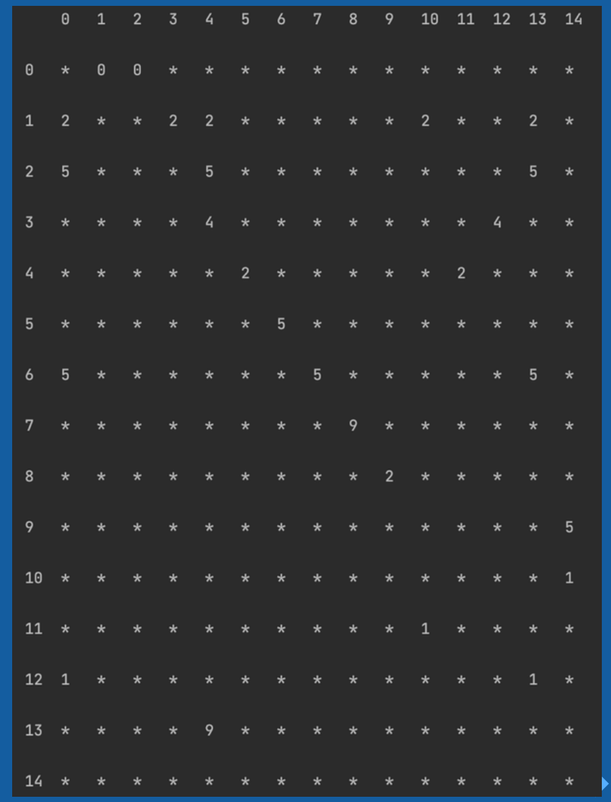
Task	Duration	Predecessors
1	1	4,5,10
2	2	7
3	3	6,10
4	4	none
5	5	none
6	6	none
7	7	none
8	9	5,7
9	9	1,2,3,8
10	10	none

2. VALUE MATRIX

```python
def print_value_matrix(data):
    for x in range(4):
        print("\n")
    nb_vertices = len(data)
    for i in range(nb_vertices+3):
        for j in range(nb_vertices+3):

            # Print the first line
            if i == 0:...



            # Print the second line (particular case)
            elif(i==1):...

            elif(i==nb_vertices+2):...

            else:...

    return
```

print_value_matrix that prints a matrix containing values:

1. It iterates through each row in the matrix and then iterates through each value in the row.
2. It prints each value with an unknown formatting pattern.

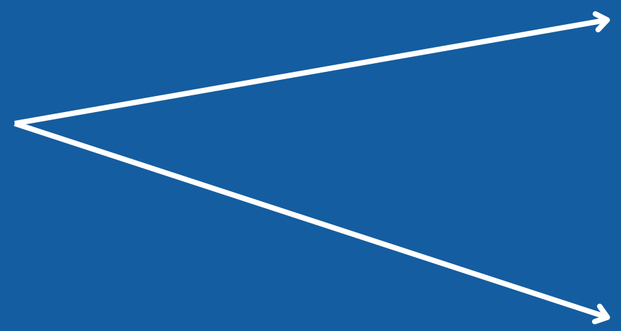


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	*	0	0	*	*	*	*	*	*	*	*	*	*	*	*
1	2	*	*	2	2	*	*	*	*	*	2	*	*	2	*
2	5	*	*	*	5	*	*	*	*	*	*	*	*	5	*
3	*	*	*	*	4	*	*	*	*	*	*	*	4	*	*
4	*	*	*	*	*	2	*	*	*	*	*	2	*	*	*
5	*	*	*	*	*	*	5	*	*	*	*	*	*	*	*
6	5	*	*	*	*	*	*	5	*	*	*	*	*	5	*
7	*	*	*	*	*	*	*	*	9	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*	2	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*	*	*	*	*	5
10	*	*	*	*	*	*	*	*	*	*	*	*	*	*	1
11	*	*	*	*	*	*	*	*	*	*	1	*	*	*	*
12	1	*	*	*	*	*	*	*	*	*	*	*	*	1	*
13	*	*	*	*	9	*	*	*	*	*	*	*	*	*	*
14	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

3. CYCLES, NEGATIVE ARCS

Is the matrix acyclic ?

```python
def is_acyclic(matrix1):
    matrix = copy.deepcopy(matrix1)
    rows_to_remove = []
    succ = []
    cycle = []
    rien = True
    while rien and matrix:

        for i, row in enumerate(matrix):
            if 'none' in row:...
            # print(rows_to_remove)

        for index in sorted(rows_to_remove, reverse=True):
            del matrix[index-1]
        # print(matrix)
        rows_to_remove.clear()

        for i in range(len(matrix)):
            if ',' in matrix[i][2]:...

            elif ',' not in matrix[i][2] and matrix[i][2] in succ:
                matrix[i][2] = 'none'

        succ.clear()
        for row in matrix:
            if 'none' in row:...
        else:
            rien = False
    if matrix:...
    else:
        print("\033[92mThe graph is acyclic\033[0m")
        return True
```



```
The graph is acyclic
```

```
The graph contains a cycle
```

is_acyclic function checks if a directed graph represented by a matrix is acyclic (has no cycles).

Negative arcs

```python
def negative_arcs(matrice):
    cpt = 1
    for i in range(len(matrice)):
        if ('-' in matrice[i][1]) or (',' in matrice[i][1]):
            cpt = -1

    if cpt == -1:
        print("\033[91mThis problem contains negative arcs\033[0m\n\n")
        return True
    else:
        print("\033[92mNo negative arcs\033[0m\n")
        return False
```

```
No negative arcs
```

negative_arcs looks if a weight matrix contains negative arcs (weights).

1. It iterates through the matrix and checks for negative values or commas within each weight, indicating a negative arc.
2. If a negative arc is found, it prints a message indicating the presence of negative arcs; otherwise, it returns True.

4. SCHEDULING

Rank

```python
def rank_func(M):
    constraint_matrix = copy.deepcopy(M)
    rank = []
    long = len(constraint_matrix)
    p = False
    old_car = []
    n = 0
    while long > 0:
        if p:
            for i in rank[n]:
                for e in range(len(constraint_matrix)):
                    if i in constraint_matrix[e][2]:...
            n += 1

        new_list = []
        for e in constraint_matrix:
            if e[2] == "none" and e[0] not in old_car:...
        rank.append(new_list)

        p = True

    rank.insert(0, ["A"])
    rank.append(["W"])

    final_matrix = []
    number_matrix = []
    rank_matrix = []
    incr = 0
    for e in rank:...

    final_matrix.append(number_matrix)
    final_matrix.append(rank_matrix)

    return final_matrix
```

The function rank_func processes a constraint matrix (M), ranking tasks based on constraints and returning a final matrix containing the original information.

Task	A	1	2	3	6	4	7	11	8	13	5	9	10	12	W
Rank	0	1	1	2	2	3	3	3	4	4	5	5	6	7	8
Duration	0	2	5	4	2	1	5	19	5	1	9	9	2	5	0
Predecessors	--	A	A	1	1,2	3	6	6	7	6,11	4,6,8	8	9	7,8,9,10	12,13,5

Earliest dates

```python
def calculate_earliest_dates(tasks_matrix):
    durations = list(map(int, tasks_matrix[2]))

    earliest_dates = [0] * len(tasks_matrix[0])

    # Create a dictionary to map task names to their indices
    task_indices = {task: i for i, task in enumerate(tasks_matrix[0])}

    # Iterate through tasks to calculate earliest start dates
    for i in range(len(tasks_matrix[0])):
        predecessors = tasks_matrix[3][i].split(',')

        # If task has predecessors, calculate earliest start date
        if predecessors[0] != '--':
            max_completion_date = max(
                [earliest_dates[task_indices[predecessor]] + durations[task_indices[predecessor]] for predecessor in
                 predecessors])
            earliest_dates[i] = max_completion_date

    tasks_matrix.append(list(map(str, earliest_dates)))
    return tasks_matrix
```

This code calculates the earliest start dates for tasks in a project schedule, considering dependencies between tasks.

Successor

```python
def create_successor(tasks_matrix):
    task = tasks_matrix[0]
    predecessors = tasks_matrix[3]

    successor_row = []

    for i in range(len(task)):
        successor_list = []
        for j in range(len(predecessors)):
            if task[i] in predecessors[j].split(','):
                successor_list.append(task[j])
        if not successor_list:
            successor_row.append('--')
        else:
            successor_row.append(','.join(successor_list))

    tasks_matrix.append(successor_row)
    return tasks_matrix
```

create_successor checks if it's a predecessor and returns a matrix containing the successor lists for all tasks.

Latest dates

```python
def calculate_latest_dates(tasks_matrix):
    task_ids = tasks_matrix[0]
    durations = list(map(int, tasks_matrix[2]))
    successors = tasks_matrix[-1]

    latest_dates = [0] * len(task_ids)

    # Find the index of the task 'W'
    w_index = task_ids.index('W')

    # Set the latest date for 'W' to its earliest date
    latest_dates[w_index] = tasks_matrix[-2][w_index]

    # Iterate through tasks from right to left
    for i in range(len(task_ids) - 2, -1, -1):
        if successors[i] == 'W':
            latest_dates[i] = int(latest_dates[-1]) - durations[i]
            continue

        successor_indices = [int(successor) for successor in successors[i].split(',')]
        min_successor_latest_date = find_latest_dates(tasks_matrix, successor_indices, latest_dates)

        latest_dates[i] = int(min_successor_latest_date) - durations[i]

    tasks_matrix.append(list(map(str, latest_dates)))
    return tasks_matrix
```

calculate_latest_dates calculates the latest allowable finish dates for each task in a project schedule.

Result:

Task	A	1	2	3	6	4	7	11	8	13	5	9	10	12	W
Rank	0	1	1	2	2	3	3	3	4	4	5	5	6	7	8
Duration	0	2	5	4	2	1	5	19	5	1	9	9	2	5	0
Predecessors	--	A	A	1	1,2	3	6	6	7	6,11	4,6,8	8	9	7,8,9,10	12,13,5
Earliest Date	0	0	0	2	5	6	7	7	12	26	17	17	26	28	33
Successors	1,2	3,6	6	4	7,11,13,5	5	8,12	13	5,9,12	W	W	10,12	12	W	--
Latest Date	0	3	0	19	5	23	7	13	12	32	24	17	26	28	33

Total float

```
def find_total_float(tasks_matrix):
    earliest_dates = tasks_matrix[4]
    latest_dates = tasks_matrix[6]
    total_float = []
    for i in range(len(earliest_dates)):
        total_float.append(int(latest_dates[i]) - int(earliest_dates[i]))

    tasks_matrix.append(list(map(str, total_float)))
    return tasks_matrix
```

find_total_float calculates the total float for tasks. The total float represents the slack time available for a task without delaying the project's overall completion.

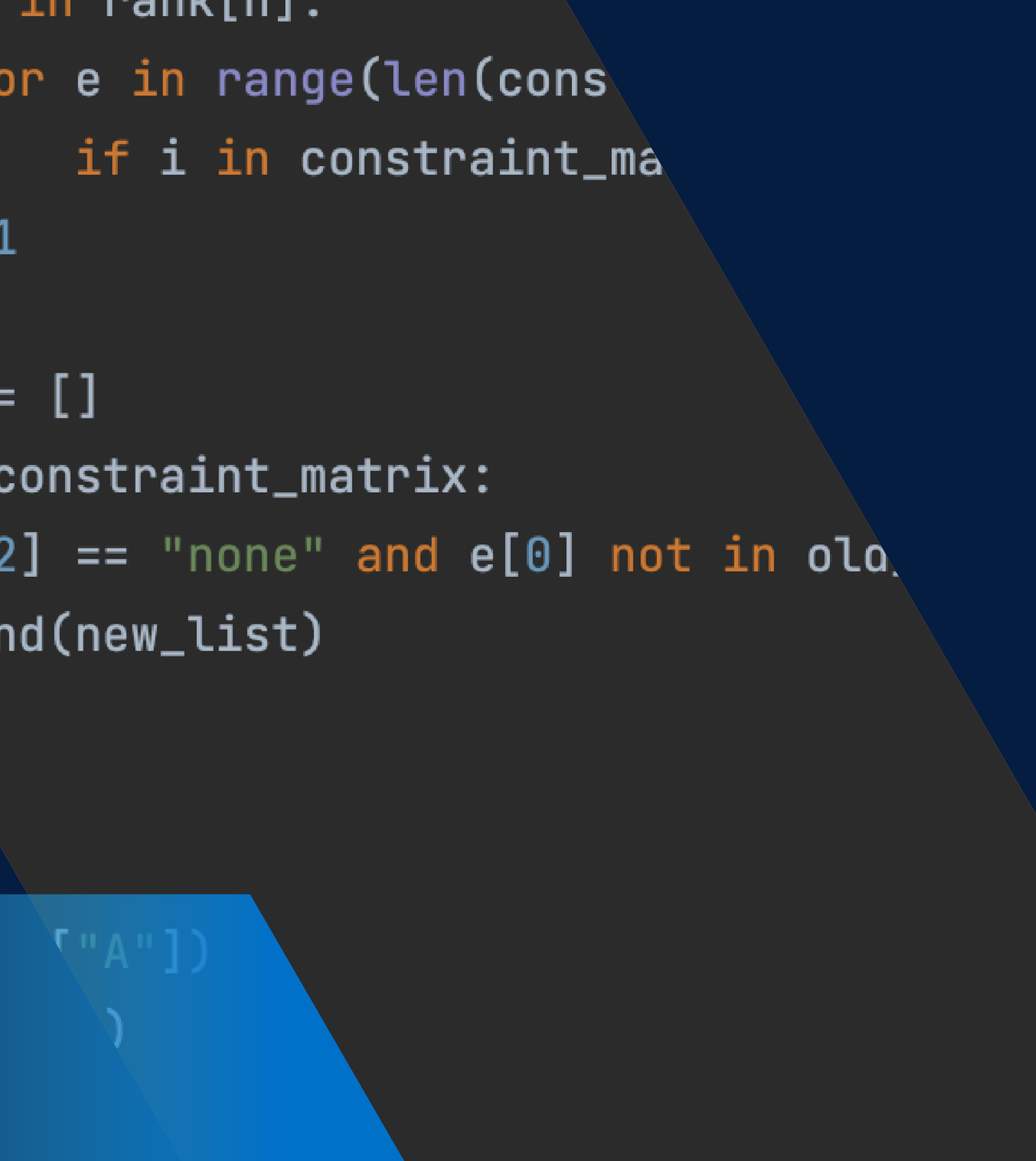
Task	A	1	2	3	6	4	7	11	8	13	5	9	10	12	W
Rank	0	1	1	2	2	3	3	3	4	4	5	5	6	7	8
Duration	0	2	5	4	2	1	5	19	5	1	9	9	2	5	0
Predecessors	--	A	A	1	1,2	3	6	6	7	6,11	4,6,8	8	9	7,8,9,10	12,13,5
Earliest Date	0	0	0	2	5	6	7	7	12	26	17	17	26	28	33
Successors	1,2	3,6	6	4	7,11,13,5	5	8,12	13	5,9,12	W	W	10,12	12	W	--
Latest Date	0	3	0	19	5	23	7	13	12	32	24	17	26	28	33
Total Float	0	3	0	17	0	17	0	6	0	6	7	0	0	0	0

Critical path

```python
def critical_path(tasks_matrix):
    tasks = tasks_matrix[0]
    total_float = tasks_matrix[-1]
    path = []
    for i in range(len(tasks)):
        if int(total_float[i]) == 0:
            path.append(tasks[i])
    print("\nThe critical path is : \n")
    for j in range(len(path)):
        print(" --> \033[91m", path[j], "\033[0m", end='')

    print()
```

```
The critical path is :

 -->  A  -->  2  -->  6  -->  7  -->  8  -->  9  -->  10  -->  12  -->  W
```

critical_path function finds the critical path. The critical path is the sequence of tasks with zero slack time, meaning any delays in these tasks will directly impact the project deadline.


in rank[n]:
or e in range(len(cons
if i in constraint_ma
1
= []
constraint_matrix:
2] == "none" and e[0] not in olo
nd(new_list)
["A"])
)

CONCLUSION