Graph Theory

Polytech Tours
2018-2019

Finishing the first part

- For `SearchChain_ts(`u_0`)` use the familly of lists `_aprec`, `_bprec`, `_nprec`, `_asucc`, `_bsucc`, `_nsucc`

```
for j in succ[i]
       the_succ = j
       ⋮
```

```
for j in range(_asucc[i],_asucc[i+1])
       the_succ = _bsucc[j]
       the_arc = _nsucc[j]
       ⋮
```

We have everything about this arc

- At the end of `SearchChain_ts(`u_0`)` we are supposed to have:
  - The lists `Predecessor` & `Successor`
  - The list of marked vertices

Finishing the first part

- Add the following lists:
  - `mu_plus=[]` will contain the arcs in the positive sense
  - `mu_minus=[]` will contain the arcs in the negative sense
- Then, create a routine `IdentifyChain(`u_0`)`
  - that will be called if there is a chain,
  - and that will use `Predecessor` and `Successor` to create the list of arcs of the chain which is split into `mu_plus` and `mu_minus`

Similarly with Successor

```
Algorithm:
        dest = Origine[u0]
        orig = Destination[u0]
        i=dest
        While i ≠ orig Do
                if Predecessor[i] ≠ -1 then
                        • k = Predecessor[i]
                        • search the number
                        of the arc (k,i) and
                        add this arc (depending on
                        the color of u0), to
                        mu_plus or to mu_minus
                        • i = Predecessor[i]
                if Successor[i] ≠ -1 then
                        • k = Successor[i]
                        • search the number
                        of the arc (i,k) ...
```

TP n°2

I. Feasible flow
   I. Graph reading & coding
   II. Some routines
   III. SearchChainColor(u_0) with colors
   IV. Feasible flow

TP n°2

TP n°2

I. Graph reading & coding

TP n°2

I. Graph reading & coding

- A graph is described in a text file in a very simple way :

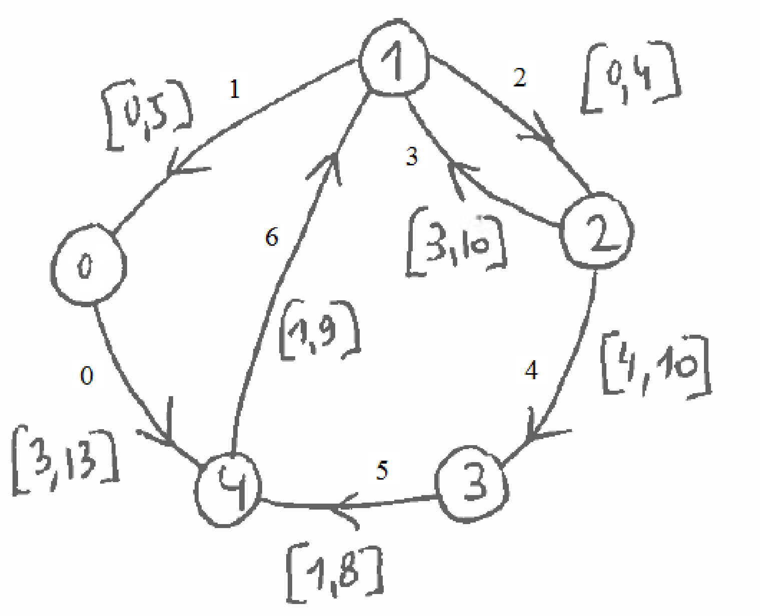
For all the arcs:

```
orig_v - tab - dest_v - tab - min_cap - tab - max_cap
```

graph_TP2.txt - Bloc-notes

Fichier Edition Format Affichage ?

0	4	3	13
1	0	0	5
1	2	0	4
2	1	3	10
2	3	4	10
3	4	1	8
4	1	1	9



TP n°2

I. Graph reading & coding

- 1st step
  - Open the file
  - Read the file
  - Close the file

```python
# ##############################
# Read the data
# ##############################

file_graph = 'graph_TP1.txt'

# Format of the file:
# For each arc :
# origine_vertex - tab - destination_vertex - tab - min_capacity - tab - max_capacity - tab - cost

# Open the file - read - close the file
TheGraph = open(file_graph,"r")
all_arcs = TheGraph.readlines()
TheGraph.close()

print (all_arcs)
```

TP n°2

I. Graph reading & coding

- 2nd step
  - Code the graph
  - We have to read '**0**\t**1**\t**5**\t**12**\n'and to extract '**0**', '**1**', '**5**', and '**12**' ⇒ we use **split**
  - We have to delete '\n' ⇒ we use **strip**

- We define four lists of size M:
  - `Origine`[u] = vertex origine of arc u
  - `Destination`[u] = vertex destination of arc u
  - `MinCapacity`[u]
  - `MaxCapacity`[u]

```python
# Fill the structures
Origine = []
Destination = []
MinCapacity = []
MaxCapacity = []
for one_arc in all_arcs:
    this_arc = one_arc.split("\t")
    orig = int(this_arc[0])
    dest = int(this_arc[1])
    mincap = int(this_arc[2])
    maxcap = int(this_arc[3].strip("\n"))
    Origine.append(orig)
    Destination.append(dest)
    MinCapacity.append(mincap)
    MaxCapacity.append(maxcap)

print('Origine=',Origine)
print('Destination=',Destination)
print('MinCapacity=',MinCapacity)
print('MaxCapacity=',MaxCapacity)
```

TP n°2

I. Graph reading & coding

- For the graph structure, we keep:
  - Origine, Destination
  - prec, succ
  - _aprec, _bprec, _nprec, _asucc, _bsucc, _nsucc

- We introduce some lists
  - `Flow=[]` of size M that will contain the flows
  - `Color=[]` of size M that will contain the colors
  - `Distance=[]` of size M that will contain the distance of the arcs to the « feasibility »
  - `mu_plus=[]` and `mu_minus=[]` will contain the arcs of the chain/cycle

TP n°2

II. Some routines

TP n°2

II. Some routines

- First routine: `UpdateColor(u)` returns the color of arc u (according to the flow value, min and max capacities)

- Second routine: `TotalDistance()` returns the total distance of the flow to the feasibility (and update the `Distance[u]` for each arc).

- Initialize the list `Flow` to `[0 for u in range(0,NbArcs)]`
- Initialize the list `Color` by calling `UpdateColor()` M times

TP n°2

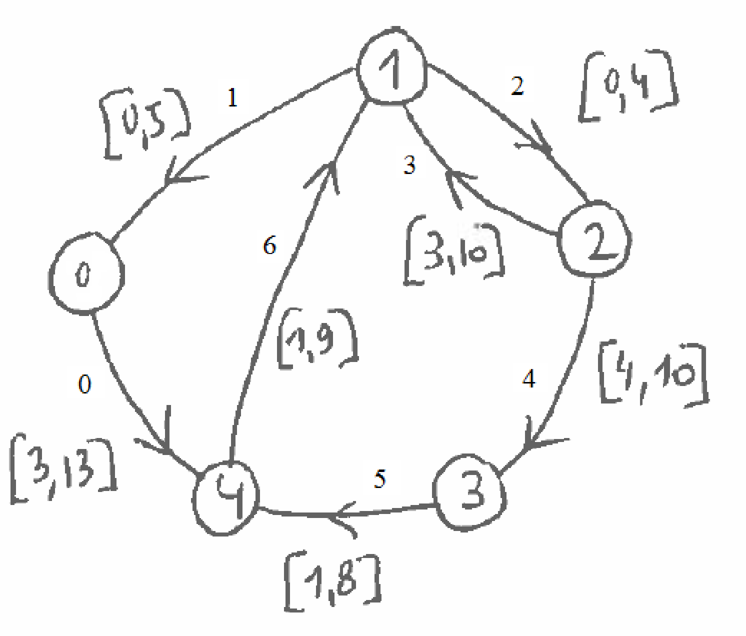
III. SearchChainColor(u_0)

TP n°2

III. SearchChainColor(u_0)

- `SearchChainColor(`u_0`)`
  - is a routine searching for a chain from `Destination[`u_0`]` to `Origine[`u_0`]` with the black arcs in the same sense, the green arcs in the opposite sense, red arcs in an arbitrary sense and without uncolored arcs.

To check the color you need the number of the arc



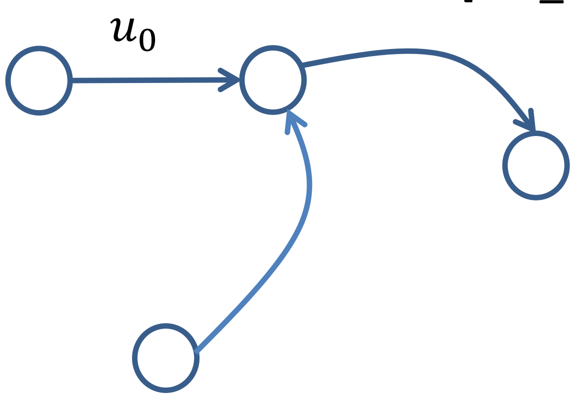
TP n°2

III. SearchChainColor(u_0)

- `SearchChainColor(`u_0`)`
  - is a routine searching for a chain from `Destination[`u_0`]` to `Origine[`u_0`]`. Put in the stack the successors if the color of the arc is red or the same as u_0 and put the predecessors if the color of the arc is red or the opposite color of u_0.
  - Use `opposite = {'b':'g','g':'b'}`

Write the algorithm
of `SearchChain_ts_Color`
Then, code it.
Give an arbitrary color to the arcs
and test it

Successor
if Color[the_arc] in [Color[u0],'r']



Predecessor
if Color[the_arc] in [opposite[Color[u0]],'r']

TP n°2

IV. Feasible flow

TP n°2

IV. Feasible flow: algorithm

- Boolean `feasible_flow` ← True
- While `TotalDistance()` >0 and `feasible_flow` Do
  - Search an arc u_0 with distance >0 (the arc with the maximum distance)
  - Initialize `Marked`, `Predecessor`, `Successor`, `mu_plus`, `mu_minus` = []
  - If `SearchChainColor(`u_0`)`
    - Identify `mu_plus` and `mu_minus`, add u_0 to `mu_plus` or `mu_minus`
    - Compute `epsilon`
    - Modify the `Flow`
    - Update the Colors of the arcs in `mu_plus+mu_minus`
  - Else
    - `Feasible_flow` ← False
  - Endif
- EndWhile

- Else
  - `Feasible_flow` ← False
  - Identify the Marked vertices (setA)
  - Identify $\omega^+(A)$ and $\omega^-(A)$
  - Show the inequality
- Endif

SUMMARIZE

- Step 1: SearchChain_ts(u0) with _aprec, _asucc…
- Step 2:
  - mu_plus, mu_minus,
  - IdentifyChain()
- Step 3:
  - MinCapacity, MaxCapacity, Flow, Color, Distance
  - UpdateColor(u)
  - TotalDistance()
- Step 4: SearchChainColor(u0)
- Step 5: FeasibleFlow()