Théorie des Graphes

Polytech Tours
2018-2019

Finishing the first part (really)

```
file_graph = 'graph_TP2.txt'
TheGraph = open(file_graph,"r")
all_arcs = TheGraph.readlines()
TheGraph.close()

# 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)

NbArcs = len(Origine)
NbVertices = max(max(Origine),max(Destination))+1
```

```
succ = [[] for j in range(0,NbVertices)]
prec = [[] for j in range(0,NbVertices)]
numsucc = [[] for j in range(0,NbVertices)]
numprec = [[] for j in range(0,NbVertices)]
for u in range(0,NbArcs):
    i = Origine[u]
    j = Destination[u]
    succ[i].append(j)
    numsucc[i].append(u)
    prec[j].append(i)
    numprec[j].append(u)
```

Finishing the first part (really)

```
_asucc = []
_bsucc = []
_nsucc = []
_isucc = 0

_aprec = []
_bprec = []
_nprec = []
_iprec = 0

for j in range(0,NbVertices):
    _asucc.append(_isucc)
    _isucc = _isucc + len(succ[j])
    _bsucc = _bsucc + succ[j]
    _nsucc = _nsucc + numsucc[j]
    _aprec.append(_iprec)
    _iprec = _iprec + len(prec[j])
    _bprec = _bprec + prec[j]
    _nprec = _nprec + numprec[j]
_asucc.append(_isucc)
_aprec.append(_iprec)
```

Finishing the first part (really)

```
def SearchChainColor(u0):
    global Marked
    print('*** search path from arc', u0,' : ',Destination[u0],' to ',Origine[u0])
    dep = Destination[u0]
    arr = Origine[u0]
    In_Stack = [False for j in range(0,NbVertices)]
    opposite = {'g':'b','b':'g'}
    List = []
    List.append(dep)
    found = False
    while (List != []) and (not found):
        i = List[0]
        Marked[i] = True
        del(List[0])
        for j in range(_asucc[i],_asucc[i+1]):
            the_succ = _bsucc[j]
            the_arc = _nsucc[j]
            if (the_succ == arr) and (Color[the_arc] in [Color[u0],'r']): found = True
            if (not In_Stack[the_succ]) and (not Marked[the_succ]) \
                and (Color[the_arc] in [Color[u0],'r']):
                List.append(the_succ)
                In_Stack[the_succ] = True
                Predecessor[the_succ] = i
        for j in range(_aprec[i],_aprec[i+1]):
            the_prec = _bprec[j]
            the_arc = _nprec[j]
            if (the_prec == arr)  and (Color[the_arc] in [opposite[Color[u0]],'r']) and the_arc != u0: found = True
            if (not In_Stack[the_prec]) and (the_arc != u0) and (not Marked[the_prec]) \
               and (Color[the_arc] in [opposite[Color[u0]],'r']):
                List.append(the_prec)
                In_Stack[the_prec] = True
                Successor[the_prec] = i
    return(found)
```

Finishing the first part (really)

```
def IdentifyChain(u0):
    #global the_chain
    global mu_plus
    global mu_minus
    global Predecessor
    global Successor
    dest = Origine[u0]
    orig = Destination[u0]
    i = dest
    while (i != orig):
        if Predecessor[i] != -1:
            k = _aprec[i]
            ind = _aprec[i]+_bprec[_aprec[i]:_aprec[i+1]].index(Predecessor[i])
            if Color[u0]=='b': mu_plus.append(_nprec[ind])
            else: mu_minus.append(_nprec[ind])
            i=Predecessor[i]
        elif Successor[i] != -1:
            k = _asucc[i]
            ind = _asucc[i]+_bsucc[_asucc[i]:_asucc[i+1]].index(Successor[i])
            if Color[u0]=='b': mu_minus.append(_nsucc[ind])
            else: mu_plus.append(_nsucc[ind])
            i=Successor[i]
    return()
```

Finishing the first part (really)

```
# ##########################################
# Some routines
# ##########################################

def UpdateColor(u):
    [redacted]
    return(col)

def TotalDistance():
    [redacted]
    return(sum(Distance))

Flow = [0 for j in range(0,NbArcs)]
Color=['u' for j in range(0,NbArcs)]
Distance = [0 for j in range(0,NbArcs)]
for u in range(0,NbArcs):
    Color[u]=UpdateColor(u)
```

TP n° 3-4

I. MinCost flow
   1. Graph reading & coding
   2. New routines
   3. MinCost flow

TP n°3-4

TP n°3-4

I. Graph reading & coding

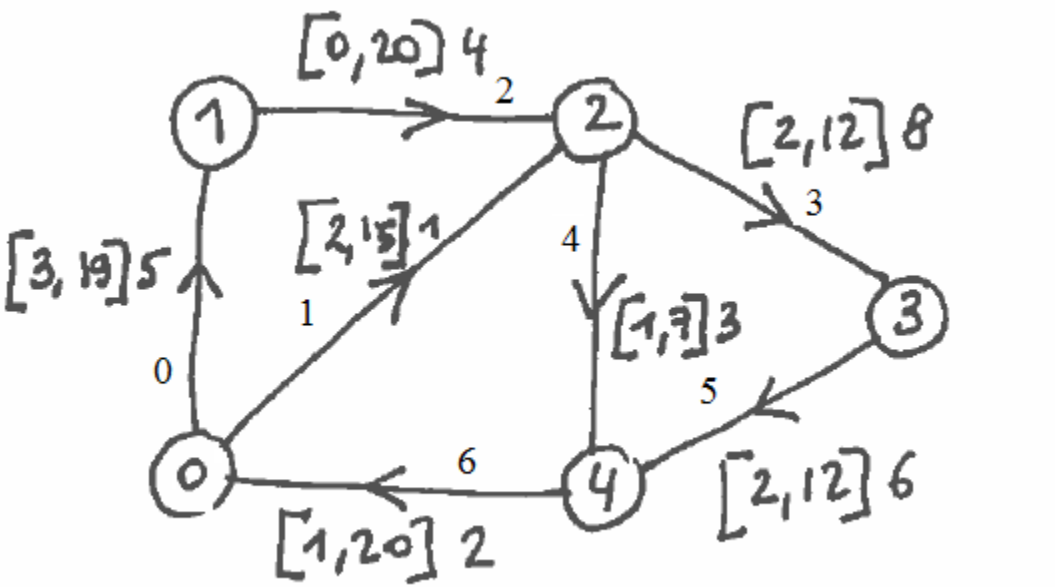
TP n°3-4

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 - tab - cost
```



graph_TP3.txt - Bloc-notes

Fichier Edition Format Affichage ?

0	1	3	19	5
0	2	2	15	1
1	2	0	20	4
2	3	2	12	8
2	4	1	7	3
3	4	2	12	6
4	0	1	20	2

TP n°3-4

I. Graph reading & coding

- We introduce one list of size M:
  - `Cost[`u`]` = cost of arc u

- We introduce the list of size M:
  - `Theta[u]` = tension of arc u

```
# Fill the structures
Origine = []
Destination = []
MinCapacity = []
MaxCapacity = []
Cost = []
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])
    cost = int(this_arc[4].strip("\n"))
    Origine.append(orig)
    Destination.append(dest)
    MinCapacity.append(mincap)
    MaxCapacity.append(maxcap)
    Cost.append(cost)

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

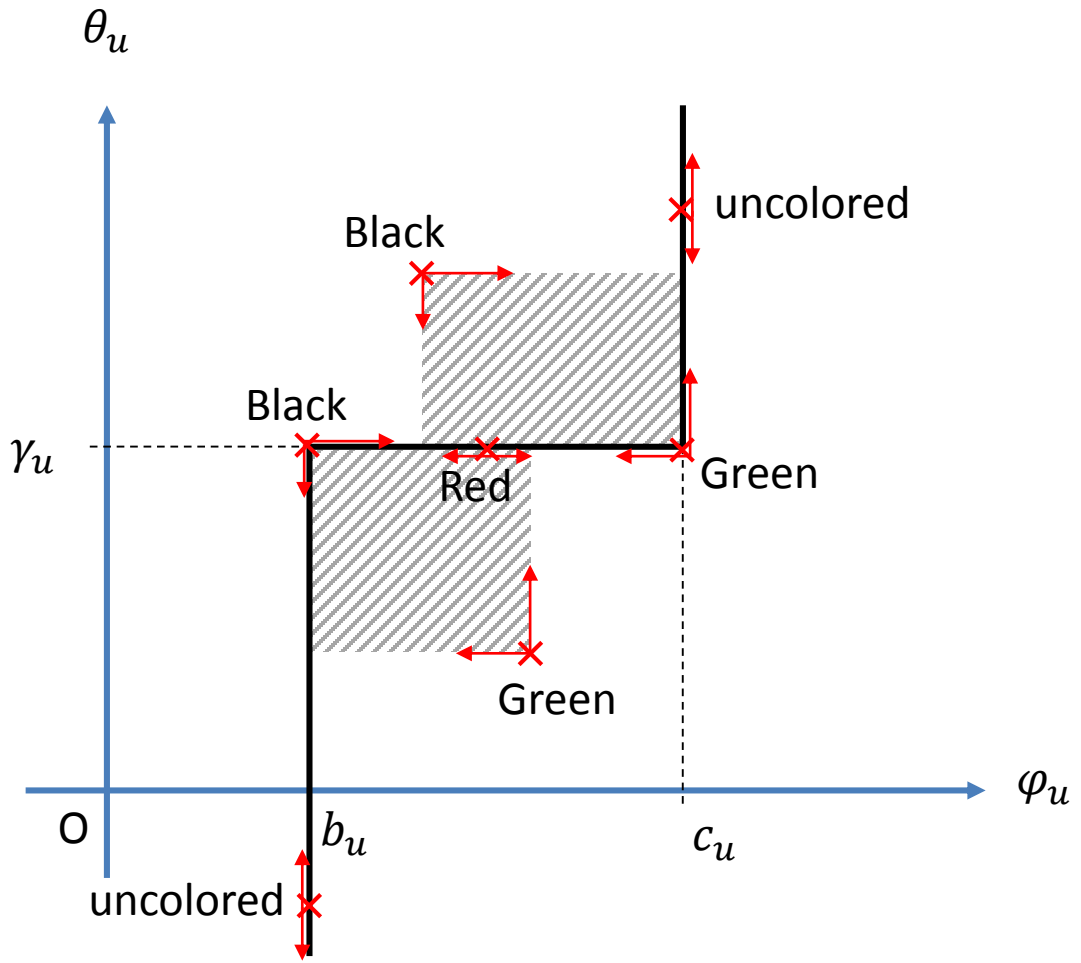
TP n°3-4

II. New routines

TP n°3-4

II. New routines

- Create the routines:
  - NewUpdateColors(u)
  - NewTotalDistance()



TP n°3-4

III. MinCost Flow: algorithm

TP n°3-4

III. MinCost Flow: algorithm

- While `NewTotalDistance()`>0 Do
  - Search an arc u_0 with distance >0 (the arc with the maximum distance)
  - Initialize `Marked`, `Predecessor`, `Successor`
  - If `SearchChainColor(`u_0`)`
    - Initialize `mu_plus` and `mu_minus`
    - `IdentifyChain(`u_0`)`, 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
    - Identify the Marked vertices (setA) according to the color of u_0
    - Identify `omega_plus` and `omega_minus`
    - Compute `epsilon`
    - Modify `Theta`
    - Update the `Colors` of the arcs in `omega_plus+omega_minus`
  - Endif
- EndWhile