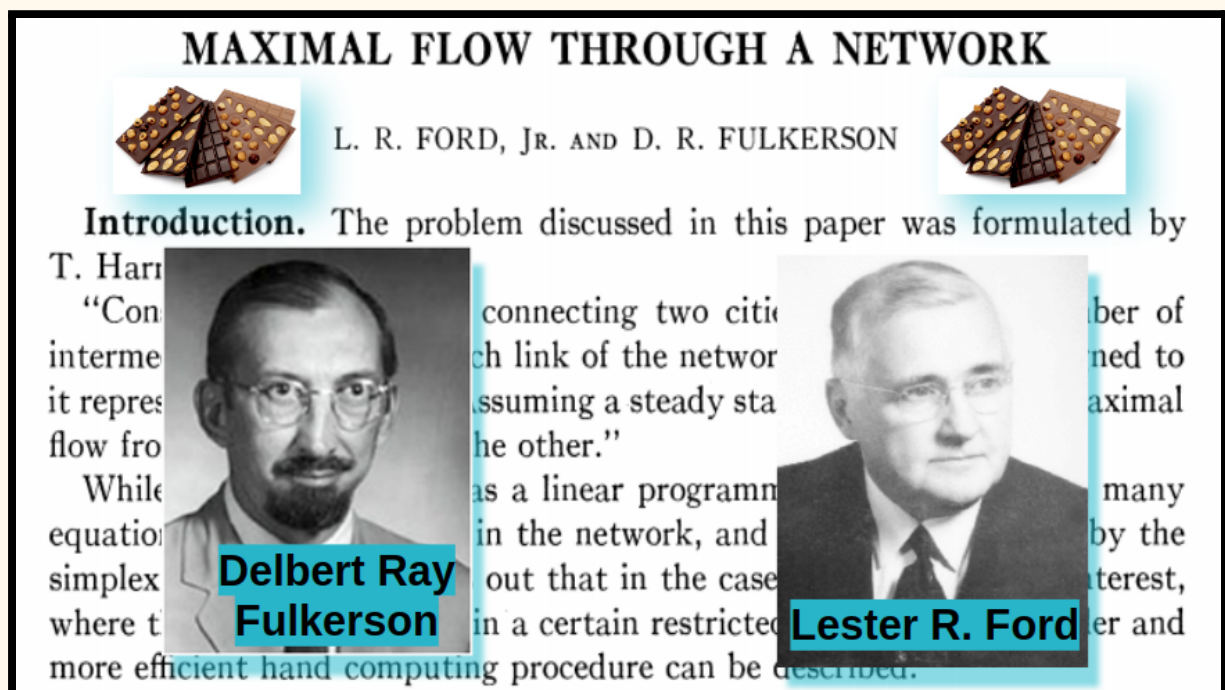


"Min Coût (Max (Choco))"

ou

L'optimisation d'un planning de
production, de **stockage** et de **transfert**
d'un **chocolatier** à **plusieurs usines**
obtenu grâce à un algorithme de flots.



D'après l'article¹ de Ford et Fulkerson sur l'algorithme du même nom et qui date de 1956 !

Objectifs. Comprendre, développer et expérimenter l'algorithme de Ford-Fulkerson sur les différentes situations proposées. Ce TD&P **noté** devrait mettre en valeur la puissance d'une modélisation d'un problème réel en un problème de flots dans un graphe. Au travers d'une application concrète, on cherche à satisfaire les consommations localisées d'un produit manufacturé selon la capacité de production, de transport, et de stockage de plusieurs usines de fabrication.

¹ Ford, L., & Fulkerson, D. (1956). Maximal Flow Through a Network. Canadian Journal of Mathematics, 8, 399-404. doi:10.4153/CJM-1956-045-5.



Rapport & Code.

Votre **code** sera accompagné d'un rapport en format **.pdf** où figurera les informations suivantes :

- les **captures d'écran** de votre terminal pour chaque question qui fait l'objet d'une réponse des algorithmes à implémenter. Ceci concerne les parties "**C++**."
- des **explications** sur vos choix de structures de données et d'algorithmes lorsqu'ils n'ont pas été pleinement définis dans ce sujet.
- les réponses aux "**Questions**."
- les "**Trace-s**." d'exécution seront reportées dans le rapport en se basant sur les exercices vus en cours et TD&P.

Un effort sur l'orthographe sera vivement apprécié. Le tout (code et rapport) sera *zippé* ou *targzé* dans un fichier "**prenom.nom.CIR2Flots.zip**" ou **.targz**. La soumission du devoir se fera à travers Microsoft Teams.

Plan.

Partie 1. Trace de l'algorithme de Ford-Fulkerson.

Partie 2. Implémentation de l'algorithme de parcours en largeur puis de l'algorithme de Ford-Fulkerson qui se servira du parcours.

Partie 3. Compréhension et modélisation de problèmes réels en graphes et recherche du flot maximum dans ces derniers. Les arcs des graphes supportent des capacités.

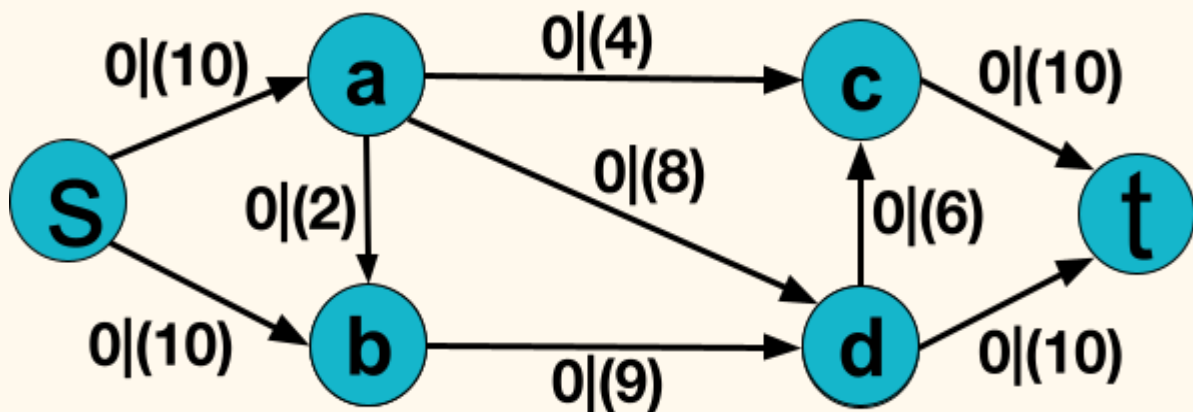
Partie 4. A vous de jouer sur l'écriture d'un algorithme qui va calculer le coût minimal d'un flot dont la valeur (i.e., le nombre d'unités) a déjà été calculée. Les arcs des graphes supportent des capacités et des coûts.

Bonus.



Partie 1 : Trace de l'algorithme de Ford-Fulkerson.

- **Trace.** On va tout d'abord s'exercer "à la main" sur un exemple simple. Déroulez l'algorithme de Ford-Fulkerson sur l'exemple ci-dessous afin de calculer le flot max entre les sommets s et t . Votre rapport doit contenir chaque étape de la trace. Une étape correspond ici à une itération de la boucle "Tant Que". A vous de choisir le type de représentation de la trace sur ce qui est proposé dans le cours. C'est pour chaque étape que les valeurs des différentes variables doivent apparaître.



Graphe d'exercice à la trace de l'algorithme de Ford-Fulkerson.

Partie 2 : Implémentation de l'algorithme de Ford-Fulkerson.

Partie 2. Étape 1. Représentation mémoire du graphe.

Nous allons tout d'abord enregistrer un graphe en mémoire avec les techniques vues en cours.

- **Question.** Quelle matrice est donnée en exemple dans le cours² pour à la fois enregistrer l'existence d'un arc mais aussi leur poids (i.e., leur coût).
- **Question.** Expliquez l'intérêt des coûts infinis dans une telle matrice.

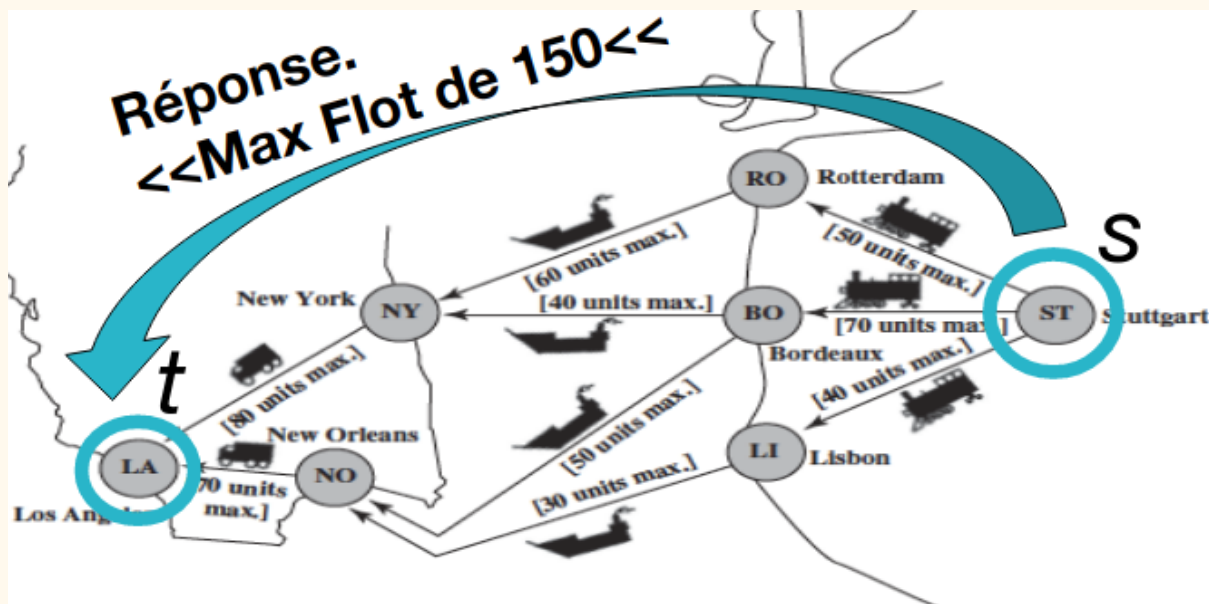
² Ici nous revenons sur la partie mémorisation des graphes (avant les flots).



Nous allons utiliser **2 matrices** du même type pour sauvegarder les informations d'un graphe. **L'une enregistrera les capacités, l'autre les coûts de chaque arc.** La matrice des coûts ne sera utilisée qu'en partie 4 : nous travaillerons d'abord sur la matrice des capacités puisque le gros du travail consistera à trouver un flot max dans les 3 premières parties. Nous évaluerons le flot en termes de coûts dans la dernière partie, mais d'abord nous cherchons à faire passer un certain nombre d'unités de flots en fonction des capacités.

- **Question.** Qu'allez-vous utiliser pour marquer le fait qu'un arc n'existe pas dans la matrice sauvegardant les capacités ?

Nous allons tout d'abord reprendre un exemple du cours, celui où l'on cherchait à acheminer un maximum d'unités de flot (ici par exemple un nombre maximum d'objets) entre la ville de Stuttgart et Los Angeles. Nous avons trouvé ensemble que le flot maximal attendu était de 150 :



Graphe d'exemple pour l'enregistrement mémoire du graphe.

- **Question.** Représentez la matrice d'adjacence **grapheEtCapacites** pour cet exemple dans votre rapport. A vous d'identifier les indices pour reconnaître les villes associées (e.g., Stuttgart == 0 et Los Angeles == 6). On met de côté la matrice des coûts pour l'instant.
- **C++.** Utilisez soit un tableau 2D (type C), soit un double <vector>, soit un <array> afin d'enregistrer votre matrice d'adjacence **grapheEtCapacites** décrite à la question précédente.



Partie 2. Étape 2. Le parcours en largeur au service de la recherche du chemin améliorant.

Nous avons déjà vu en cours que l'efficacité de l'algorithme de Ford-Fulkerson dépendait principalement de la recherche du **chemin améliorant**. Un tel algorithme peut-être basé sur un algorithme de parcours, et c'est, comme souvent, le parcours en largeur qui sera implémenté ici. Rappel. Diapositive du cours sur le chemin améliorant :

3. Graphe. 3.14. Les problèmes de flots. Définitions.

Chemin Améliorant. Un chemin améliorant est un chemin élémentaire d'une source s vers un puits t dans le graphe résiduel.

Chaque arc d'un chemin améliorant a une capacité résiduelle strictement positive -sinon l'arc disparaît.

- **C++.** Débutez l'écriture d'une fonction `parcoursLargeur` qui prendra comme paramètres :
 - un graphe représenté par une matrice du type de `grapheEtCapacites`,
 - un indice représentant le noeud de départ s ,
 - un indice représentant le noeud d'arrivée t ,
 - un vecteur (tableau 1D, vector ou array) `cheminAmeliorant` qui comme son nom l'indique sauvegardera le chemin améliorant. Ceci étant, prenons le temps d'expliquer son fonctionnement "à rebrousse-poil" :
 - Lorsque `cheminAmeliorant` a été trouvé après le parcours en largeur, il est possible de reconstituer le chemin sommet par sommet en partant du puits t vers la source s . En effet, `cheminAmeliorant[t]` nous renvoie le prédécesseur de t et nous verrons un peu plus bas que nous allons initialiser `cheminAmeliorant[s]` à `-1` puisque s n'a pas de prédécesseur.

Les variables locales de la fonction sont les suivantes :

- `bool sommetVisite[nbSommets]`
 - ce tableau est à l'image de la coloration des sommets en bleu et en blanc pour signifier qu'un sommet a été traité ou pas. Ici, la couleur bleue correspond à une valeur `true` et la couleur blanche à `false`. L'ensemble du tableau sera initialisé à `false`.
- `queue<int> file`
 - cette `file`³ est la structure principale du parcours en largeur. Vous pourrez utiliser les méthodes qui lui sont propres, par exemple :
 - `file.empty()`
 - vérifie si la file n'est pas vide.

³ N'oubliez pas d'inclure la bonne library (cf. cours ou documentation C++).



- `file.front()`
 - renvoie le sommet enfilé en tête.
- `file.pop()`
 - supprime le sommet enfilé en tête.
- `file.push(v)`
 - enfile le sommet d'indice v dans la file.

Les instructions d'initialisation sont les suivantes :

- On enfile le sommet de départ s et on indique qu'il a été visité par :
 - `sommetVisite[s] = true`
 - Seul s n'aura pas de précédent dans le chemin améliorant :
 - `cheminAmeliorant[s] = -1`
- **C++.** Nous allons maintenant nous attaquer à la boucle principale "Tant Que" de l'algorithme. Développez la, en vous basant sur le pseudo code suivant :

Tant Que la file n'est pas vide **Faire**

$u = \text{Defiler}()$; // e.g., sommet "défilé" de la file⁴

Pour tous les sommets v du graphe **Faire**

Si v n'a pas été visité et que l'arc $(u > v)$ existe⁵ **Faire**

Enfiler (v) ;

`cheminAmeliorant[v] = u` ;

`sommetVisite[v] = true` ;

Fin Si

Fin Pour

Fin Tant Que

- **C++.** Implémentez le retour de la fonction `parcoursLargeur`.
Note importante. La fonction `parcoursLargeur` se termine en renvoyant un **booléen** sur l'existence même d'un chemin reliant s à t (et donc d'un chemin améliorant). Si t n'a pas été visité au départ de s dans la boucle Tant Que, il n'y a plus la capacité suffisante pour que le graphe soit connexe.

Partie 2. Étape 3. L'algorithme de Ford-Fulkerson.

Nous avons donc développé `parcoursLargeur` qui est une implémentation du parcours en largeur adaptée pour renseigner de l'existence d'un chemin améliorant reliant un noeud s à un noeud t . Nous allons maintenant développer l'algorithme de Ford-Fulkerson que l'on notera `fordFulkerson`.

⁴ Ici cela consiste à la fois à renvoyer la valeur "first in" de la file mais aussi de l'effacer.

⁵ Nous verrons plus tard que l'existence ici dépend du graphe résiduel soit celui passé en paramètre



- **C++**. Débutez l'écriture de `fordFulkerson` en lui donnant les paramètres suivants :
 - Une matrice d'adjacence du type de `grapheEtCapacites`,
 - `int s, t`
 - les indices des deux nœuds.

Les variables locales de la fonction sont les suivantes :

- `int u, v`
 - deux indices de sommets utilisés tout le long de l'algorithme.
- `grapheResiduel`
 - matrice du type et de la taille de `grapheEtCapacites`. Comme son nom l'indique, c'est la matrice du graphe résiduel.

En plus de sa déclaration, recopiez les valeurs de `grapheEtCapacites` dans cette matrice en guise d'initialisation. Elle sera donc modifiée au cours de l'exécution.

- `cheminAmeliorant[nbSommets]`
 - c'est la même structure vue dans le parcours en largeur.
- `int max_flow`
 - cette variable qui doit être initialisée à 0 contiendra le flot max courant. C'est aussi l'entier renvoyé par la fonction `fordFulkerson`.

- **C++**. Terminer l'implémentation de `fordFulkerson`. La dernière et principale partie de l'algorithme concerne la boucle Tant Que dont le pseudo-code est donné ci-dessous :

```
Tant Que parcoursLargeur renvoie un booléen true Faire
    ameliorationFlot =  $\infty$ ;
    Pour chaque noeud v du chemin améliorant de t à s Faire
        u = cheminAmeliorant[v];
        ameliorationFlot = min(ameliorationFlot, grapheResiduel[u][v]);
    Fin Pour
    Pour chaque noeud v du chemin améliorant de t à s Faire
        u = cheminAmeliorant[v];
        grapheResiduel[u][v] -= ameliorationFlot;
        grapheResiduel[v][u] += ameliorationFlot;
    Fin Pour
Fin Tant Que
```

Partie 2. Étape 4. Testez votre programme.

- **C++**. Joignez les différentes parties de code développées en Partie 2 de façon à retrouver un flot max de 150. Reportez la capture d'écran associée au résultat obtenu à l'exécution de votre code.

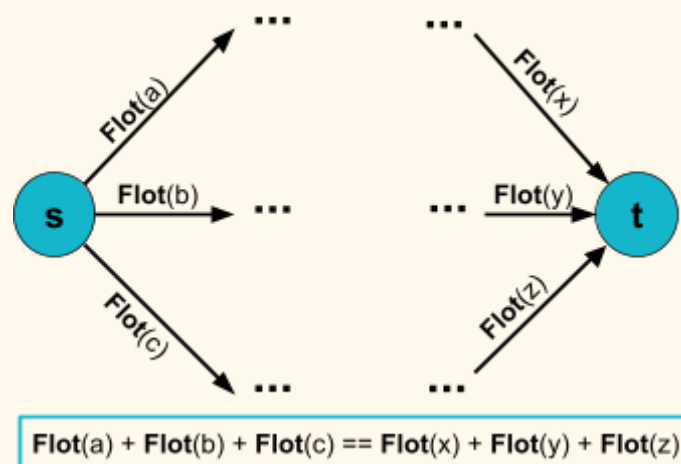


Partie 3 : Modèle et algorithme adaptés à un cas concret : optimisation de la production de chocolat.

Partie 3. Étape 1. Production libre et satisfaction de la demande.

Le calcul du flot maximum que l'on passe entre deux sommets d'un graphe peut servir à de multiples situations notamment lors de l'établissement d'un planning de production d'une entreprise qui fabrique des produits dont on notera chaque unité **P**.

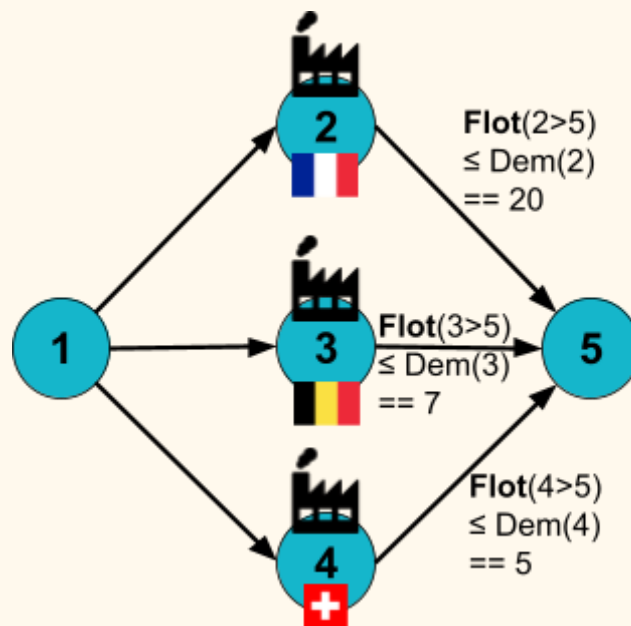
On va se servir de l'algorithme de Ford-Fulkerson pour déterminer le flot maximum tout en se servant d'une propriété à la fois simple et importante : la quantité de flot sortant du noeud source et celle entrant dans le noeud destination sont égales au flot max déterminé par l'algorithme. La figure suivante le schématise :



Règles sur les flots : il y a autant d'unités de flots qui sortent de la source *s* que d'unités qui entrent dans le puits *t*.

Imaginez avoir la responsabilité d'une société disposant de 3 usines produisant des tonnes de tablettes de chocolat⁶, chacune dans un pays différent. Pour chaque pays *i*, il y a une consommation (demande) associée **Dem(*i*)**. Dans l'exemple ci-dessous, on demande à l'usine Belge de fournir 7 unités, la française 20 unités et la Suisse 5 unités. **On considère ici que chaque unité de flot correspond à une unité de chocolat.**




⁶ La production annuelle d'un pays comme la Belgique est de plus de 600 000 tonnes. Nous prenons ici l'exemple simple qu'est le chocolat, mais vous aurez compris qu'on aurait pu poser le même problème avec des processeurs, des vaccins ou tout autre produit.



Planning de production pour une demande donnée.

On peut comprendre alors que, pour satisfaire une consommation/demande, il suffit d'exprimer cette dernière comme la capacité d'un arc partant de l'usine (sommets 2, 3, ou 4 dans la figure ci-dessus) vers le noeud destination (sommet 5) et d'appliquer Ford-Fulkerson. Ce dernier va chercher le flot maximal, et donc satisfaire toute la demande puisque les 3 capacités $\text{Dem}(i)$ limitant le nombre d'unités de flot correspondent exactement à la demande des 3 pays.

La demande locale est donnée ci-dessous :

Usine			
Demande Locale (en unité de P) Dem	20	7	5

Demande locale selon les 3 pays.

On pourra appeler '**arcs de production**' les arcs $(1>2)$, $(1>3)$ et $(1>4)$ tandis que $(2>5)$, $(3>5)$ et $(4>5)$ sont nommés '**arcs de demande**'.




- **C++.** Représentez ce nouveau graphe en mémoire et liez-le au code écrit dans la partie précédente en y incluant la demande. A vous de réfléchir à quoi faire pour modéliser les arcs de production avec les différentes techniques vues en cours, puisqu'il n'y a pas, pour l'instant, de limitation sur la capacité de production.
 - Exécutez alors Ford-Fulkerson sur ce graphe, puis faites ressortir le flot max. Faites une capture d'écran du résultat de votre sortie terminal (IDE ou console) pour l'insérer dans votre rapport.



Partie 3. Étape 2. Production limitée et satisfaction de la demande.

La fabrication de chocolat est cette fois soumise à une capacité de production. Si i est une usine, elle peut produire au maximum $\text{CapProd}(i)$ unités.

- **Représentation de Graphe.** A partir de la situation précédente, mettez à jour le graphe⁷ donné plus haut en y ajoutant les capacités de production du tableau ci-dessous. Réfléchissez à l'endroit où interviennent ces capacités. Reportez votre dessin dans votre rapport. Vous êtes libres du choix de l'outil pour le dessiner.

Usine			
Capacité de Production (unités) CapProd	25	10	8

Capacité de production selon les 3 pays.

- **C++.** Adaptez votre code avec les capacités de production données par le tableau précédent, exécutez le, et enfin reportez les captures d'écran du résultat obtenu.

Partie 3. Étape 3. Production et transfert limités et satisfaction de la demande.

On ajoute maintenant la possibilité de transférer les unités de chocolat entre les usines une fois celles-ci produites. Les transferts d'unité de chocolat peuvent être considérés comme **instantanés** et suivant le processus de production.

Les camions ou autres moyens de transport sont en nombre limité, on considère une capacité sur ces transferts que l'on notera CapTrans . Ainsi, si i et j sont deux usines, la capacité de transfert de l'usine i vers l'usine j est notée $\text{CapTrans}(i > j)$.

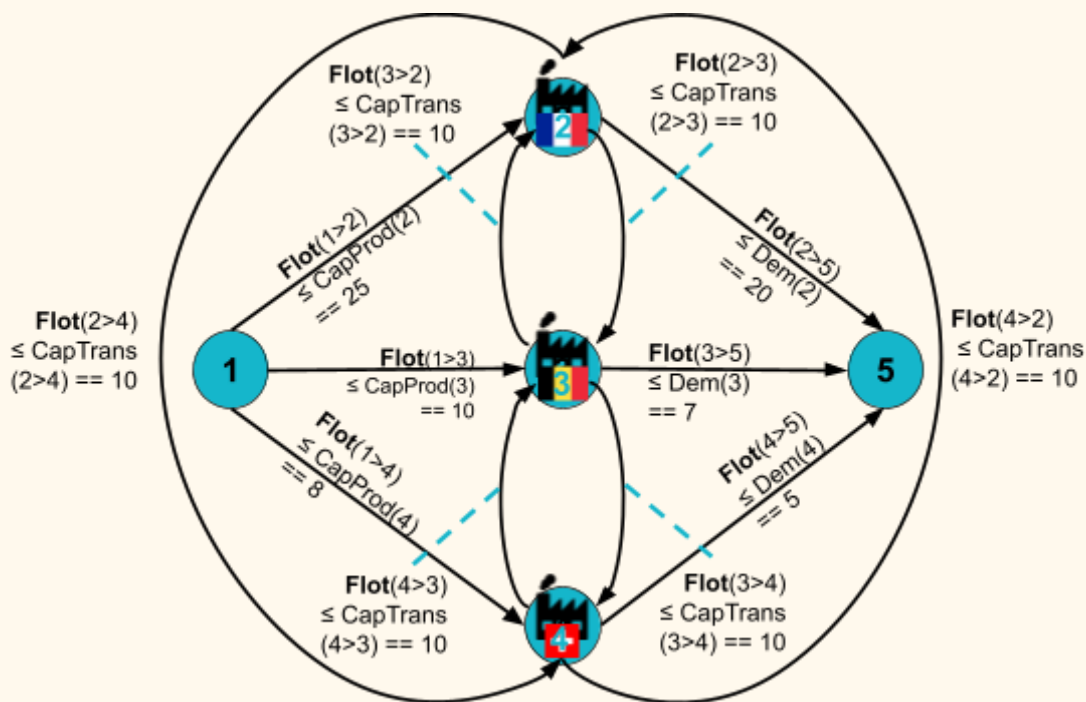
Dans le graphe ci-dessous, les 5 produits attendus en Suisse ($\text{Dem}(4) == 5$) peuvent être fabriqués en France en plus de sa propre production locale ($\text{CapProd}(2) == 25$, $\text{Dem}(2) == 20$, et $\text{Dem}(4) == 5$; on respecte bien la capacité de production de l'usine française puisque $20 + 5 \leq 25$). De plus, les 5 produits à destination de la Suisse peuvent être acheminés sans

⁷ Dans le code C++ à rendre, mettez en commentaire les situations précédentes, faites d'abord un copier-coller de celles-ci afin d'implémenter chaque évolution.



problème depuis la France par l'arc (4>2) puisque l'on vérifie l'inégalité $5 \leq \text{CapTrans}(2>4) = 10^8$.

Remarquons qu'il est possible de faire plusieurs transferts avec les mêmes unités de chocolat. Par exemple, on pourrait très bien avoir n unités produites en Belgique puis transférées en France (si $n \leq \text{CapTrans}(3>2)$), en enfin une partie $m \leq n$ de ces unités est acheminée vers la Suisse (si $m \leq \text{CapTrans}(2>4)$). Il est aussi envisageable qu'en parallèle à ces transferts, d'autres unités soient directement transférées de la Belgique à la Suisse ($\text{CapTrans}(3>4)$).



Planning : transferts et production limités.

Les capacités de transferts sont répertoriées ci-dessous (vous trouverez les autres capacités sur le graphe) :




Capacité \ arcs	(2>3)	(3>2)	(2>4)	(4>2)	(3>4)	(4>3)
Capacité de Transferts (unités) CapaTrans	8	8	12	12	3	3

Planning : transferts et production limités.

⁸ Cet exemple est diplomatiquement risqué, le transfert de chocolat dans l'autre sens aurait vraisemblablement été plus proche de la réalité.



Depuis la situation précédente, la capacité de production reste la même mais la demande française a augmenté :

Usine			
Demande Locale (en unité de P) Dem	30	7	5

Nouvelle demande locale selon les 3 pays.

- **C++**. Adaptez votre code précédent à ces capacités de transfert et cette nouvelle demande, exécutez le, et enfin reportez les captures d'écran du résultat obtenu dans votre rendu.

Partie 3. Étape 4. Production, transfert et stockage limités et satisfaction de la demande.

Nous allons maintenant considérer plusieurs périodes de temps afin de pouvoir stocker les produits d'une période à l'autre et ceci dans chaque pays. Cela permet de nouvelles possibilités pour satisfaire la demande.

Reprenons le même exemple avec nos trois usines mais cette fois avec la possibilité de stocker des d'unités de chocolat produites. Nous allons voir comment représenter ces périodes. Pour cela, "clonons" chaque sommet "usine" de façon à avoir un sommet par pays et par période de temps. Si on considère deux périodes de temps **t1** et **t2**, nous avons les sommets suivants :

- (2.1) et (2.2) : respectivement l'usine en France pour les périodes 1 et 2 ;
- (3.1) et (3.2) : respectivement l'usine en Belgique pour les périodes 1 et 2 ;
- (4.1) et (4.2) : respectivement l'usine en Suisse pour les périodes 1 et 2.




Chacun de ces nœuds dispose de capacités (production, transfert et stockage) qui leur sont propres selon la période de temps. On se sert des arcs reliés à ces nœuds pour modéliser le fait que le flot (i.e., les unités de chocolat) est soumis à toutes ces capacités. Il y a donc une capacité de production bien spécifique à chaque usine sur chaque période, et nous avons la même chose pour le transfert et le stockage. De la même manière, il existe une demande pour chaque pays à chaque période. On peut par exemple imaginer que la Belgique produit du chocolat en période 1 pour le vendre en France en période 2 du moment où le stockage et le transfert associés soient de capacité suffisante (on peut imaginer deux cas simples pour cet exemple : du stockage en Belgique puis du transfert en France ou du transfert en France puis du stockage dans ce même pays ; on peut également faire appel à la Suisse comme pays "intermédiaire").



On considère le stockage comme un processus démarrant après la production et les transferts de la période donnée. Ceci devrait se faire de manière transparente si vous respectez les caractéristiques des graphes présentés ici.

Si i est une usine, $t1$ et $t2$ deux périodes de stockage consécutives, le stockage d'unité de chocolat de la période $t1$ à la période $t2$ de l'usine i consomme une unité de capacité notée $CapStock(i, t1)$.

Le nombre d'arc augmente selon le nombre de périodes. Les nouvelles capacités de production, de transfert et de stockage, ainsi que la nouvelle demande, sont données dans la série de tableaux ci-dessous.

Usine & Périodes	 t1 & t2		 t1 & t2		 t1 & t2	
Capacité de Production (unités) CapProd	25	15	10	5	5	8

Capacités de production.


Capacité \ arcs	(2>3) t1 & t2		(3>2) t1 & t2		(2>4) t1 & t2		(4>2) t1 & t2		(3>4) t1 & t2		(4>3) t1 & t2	
Capacité de Transfert (unités) CapaTrans	10	5	4	8	10	5	4	7	10	5	10	5

Capacités de transferts.

Capacité \ arcs	(2.1>2.2)	(3.1>3.2)	(4.1>4.2)
Capacité de Stockage (unités) CapaStock	15	8	7

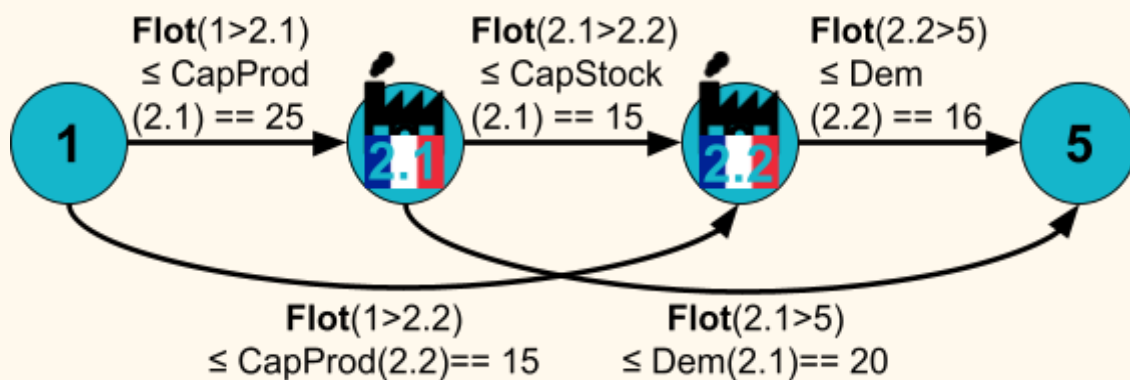
Capacités de stockage.



Usine & Périodes	 t1 & t2		 t1 & t2		 t1 & t2	
Demande Locale (en unité de P) Dem	14	19	3	10	7	5

Demande attendue.

- **Représentation de Graphe.** En vous aidant de la figure ci-dessous qui ne représente que les capacités de production et de **stockage** d'un seul pays, dessinez le graphe "global" pour toutes les capacités (production, stockage et transfert), ainsi que la demande. **Vous pouvez vous limiter à deux pays que vous aurez choisis.**⁹



Représentation des stockages.

- **C++.** Adaptez votre code pour le problème avec toutes les capacités (production, transfert et stockage) appliquées aux 3 pays en reprenant les valeurs des tableaux ci-dessus. Exécutez ensuite votre programme et reportez les captures d'écran du résultat obtenu.

⁹ Comme indiqué dans les premières questions, il est attendu d'utiliser un logiciel de dessin (Libre Office Draw, Office et ses composants pour dessins etc.).



Partie 4 : Pour aller plus loin

Min (Coût (Max (Flot))

Nous venons de voir comment calculer un plan de fabrication sur plusieurs usines en fonction de capacités de production, de transfert et de stockage, le tout sur plusieurs périodes de temps dans le but de satisfaire une demande. Une fois que le routage des différents flots traversant le graphe a été trouvé, nous allons maintenant calculer le coût de cette fabrication. Cette partie est relativement ouverte, les différents algos et codes demandés sont moins détaillés. Si vous êtes arrivés jusque là, vous avez fait le minimum attendu.

Supposez maintenant que la demande soit bien inférieure aux capacités de production (vous pouvez considérer que toutes les capacités données jusque-là ont doublé). On peut alors imaginer assez facilement qu'il existe plusieurs solutions possibles satisfaisant toute la demande, autrement dit, qu'il existe plusieurs plannings de production nous donnant le même maximum d'unités de flot allant de la source au puits (soit de *s* à *t* ou de **1** à **5** dans les exemples précédents).

- **Question.** En répondant à l'aide de quelques phrases, expliquez pourquoi il n'existerait qu'une seule solution lorsque la somme de la demande est égale à la somme des capacités de production.

Lorsqu'il existe plusieurs solutions satisfaisant toute la demande, il est intéressant de calculer le planning de coût minimal à partir du flot maximal déjà calculé. On peut alors partir à la recherche du *Min (Coût (Max (Flot))*. On va d'abord générer ces coûts, puis calculer le coût total d'une solution (i.e. le coût d'un planning qui est selon les routes prises par les unités de flot) et enfin tenter de développer un algorithme qui trouvera la solution la moins chère.

- **C++.** Génération des coûts. On va tout d'abord chercher à générer l'ensemble des coûts $c(i, j)$, $i, j \in X$, pour tous les arcs qui ne sont pas des arcs de demande¹⁰ :
 - les coûts de stockage,
 - les coûts de transfert,
 - les coûts de production.

Générez des coûts différents pour chaque type d'arc (production, transfert et stockage) et cela pour chacun des pays en vous aidant de la génération de nombres pseudo-aléatoires. Ainsi, servez-vous d'une matrice de la même taille que **grapheEtCapacites**, mais cette fois pour sauvegarder les coûts générés. **Relever le graphe des coûts que vous obtenez dans votre rapport.**

¹⁰ A vous de comprendre quelle valeur utiliser pour ces arcs de la demande.



- **C++.** **Partie libre.** Modifiez votre code de l'algorithme de Ford Fulkerson de façon à pouvoir calculer le coût d'une solution. Ainsi, il faut trouver ici un moyen de déduire le parcours des unités de flots.

Ensuite, trouvez une manière de calculer le coût total pour faire la somme des produits des coûts de chaque arc par le nombre des unités de flots qui le traverse. On peut grossièrement résumer le coût total d'une solution par :

$$\text{Coût Total} = \sum_{(i,j) \in E} c(i,j).nbUnitésDeFlot(i,j)$$

Relevez dans votre rapport le coût obtenu pour le graphe des coûts que vous avez générés dans la question précédente.

- **Algo¹¹.** **Partie libre.** Imaginez, puis écrive l'algorithme cherchant : soit le planning le moins coûteux, soit un des moins coûteux pour un flot max déjà calculé ; ceci correspond respectivement à écrire soit une méthode exacte, soit une méthode approchée¹². Que vous fassiez le premier ou le second choix, prenez le temps de bien expliquer votre algorithme dans votre rapport.
- **C++.** **Partie libre.** Développez votre idée d'algorithme et chercher, au moins, un planning de coût faible. Reportez votre résultat dans votre rapport.
- **Question.** Vous disposez donc d'un programme déroulant l'algorithme de Ford Fulkerson sur un graphe afin de calculer le flot maximal. Supposez maintenant que vous avez besoin de **savoir au plus vite s'il existe un chemin entre deux points** sans se préoccuper des vraies capacités du graphe. Ne voulant pas implémenter de nouveaux algorithmes, et n'ayant la main que sur les entrées/*inputs* du programme (i.e., les éléments définissant le graphe), expliquez qu'est-ce que vous feriez pour obtenir l'information sur l'existence d'un tel chemin.
- **Question.** Supposez disposer du programme calculant le coût minimal du passage d'une quantité de flot entre les sommets s et t qui est passée en paramètre (qu'importe que la quantité soit maximale ou pas). Vous avez besoin de savoir **quel est le plus court chemin entre s et t**. Disposant d'un graphe tel que ceux utilisés dans ce sujet, que feriez-vous pour obtenir ce plus court chemin ? Aide : on peut imaginer que les coûts unitaires sont alors des distances.

¹¹ Si vous êtes arrivé·e·s jusque là en répondant bien aux différents points précédents, c'est déjà **un très bon travail** !

¹² Comme nous l'avons vu en cours, la méthode exacte donnera systématiquement, si le temps le permet, une des (ou la) solutions optimales (ici de coût minimum) alors que la méthode approchée (appelée heuristique) cherchera une bonne solution et parfois la meilleure mais sans le savoir.



Bonus.

- a. Prendre en main une bibliothèque spécialisée dans les graphes. Utilisez la pour représenter les résultats obtenus en générant le “code Graphviz” à partir de votre programme. GraphViz est souvent utilisé en théorie des graphes (c’est d’ailleurs celle-ci qui dessine les diagrammes de classes dans les documentations Doxygen). Très facile à utiliser sous Linux, elle peut également l’être avec Visual Studio :

- i. <https://marketplace.visualstudio.com/items?itemName=joaompinto.vscode-graphviz>

Il est également possible d’utiliser GraphViz dans une version en ligne une fois le graphe écrit en mode texte :

- ii. <http://graphviz.it/>
iii. <http://www.webgraphviz.com/>
iv. <https://dreampuf.github.io/GraphvizOnline/>

Au passage, vous avez la possibilité d’avoir Microsoft Visio avec votre compte JUNIA :

<https://portal.azure.com/#blade/Microsoft Azure Education/EducationMenuBlade/software>. En plus de pouvoir générer des diagrammes automatiquement à partir de votre code C++ (pour ceux qui connaissent : diagramme de classes et autres diagrammes UML), VISIO permet de dessiner à la main des graphes assez facilement.

- b. Imaginez que les coûts de production sont la somme d’un coût variable (celui considéré jusque-là et que l’on multiplie par le nombre d’unités produites) et d’un coup fixe. Ce coût fixe est souvent appelé coût de “Setup”, il peut à la fois représenter le personnel ou encore les machines dont le coût de mise en marche est significatif. Ces coûts fixes n’ont pas d’incidence sur le calcul du flots maximum mais en a bien plus sur la minimisation du coût global. Adaptez vos algorithmes pour prendre en compte les coûts fixes : ces derniers sont compris dans le coût global dès lors qu’au moins une unité est fabriquée. Dans la littérature scientifique, ces problèmes sont appelés *Lot-Sizing*.
- c. Il existe une extension aux flots particulièrement intéressante : les multi-flots (*eng. Multicommodity flow*). Pour notre problème de planning de production/transferts/stockage, essayez de comprendre ce que sont ces multi-flots et de réfléchir à comment l’adapter à une production de produits de types différents soumis à une même capacité¹³. Il vous est juste demandé ici de rédiger un texte montrant ce que vous avez compris après avoir fait quelque recherche tout en donnant un exemple concret. Par exemple, pour les capacités de transferts, un même camion -et donc une même capacité de transfert- allant d’une usine 1 à une usine 2 peut contenir plusieurs types de produits. Présentation Wikipédia des multi-flots :

- i. https://en.wikipedia.org/wiki/Multi-commodity_flow_problem

Bon courage.

¹³ Jusqu’ici nous n’avons considéré qu’un type de produit représenté par une unité. Dans la version multi-produits, on consomme et paie l’unité selon le type de produit alors que les capacités disponibles sont globales.