Numéro d'anonymat :

UE d'algorithmique (LU3IN003).
Licence d'informatique.

Examen du 8 janvier 2024.

Seule une feuille A4 portant sur les cours et les TD est autorisée, tout autre document est interdit. Téléphones portables éteints et rangés dans vos sacs. Le barème est indicatif et est susceptible d'être modifié. Toutes les réponses doivent être justifiées.

Exercice 1 (4 points)

Répondez aux questions à choix multiples suivantes en cochant la ou les cases correspondant aux bonnes réponses. Chaque bonne réponse aux questions 1 à 4 apportera 0.5 point, tandis que chaque mauvaise réponse retranchera 0.25 point à votre note. Chaque bonne réponse aux questions 5 et 6 apportera 1 point, tandis que chaque mauvaise réponse retranchera 0.5 point à votre note. La note de l'exercice sera le maximum entre 0 et la somme des points obtenus.

Questions	**Réponses**
1. Quel algorithme parmi les suivants est de type "diviser pour régner" ?	☐ algorithme de Huffman ☐ tri fusion ☐ ordonnancement d'intervalles pondérés ☐ algorithme de Dijkstra
2. Parmi les algorithmes suivants, lequel n'est pas basé sur un parcours ?	☐ parcours en largeur ☐ algorithme de Prim ☐ algorithme de Kruskal ☐ algorithme de Dijkstra
3. Quel algorithme pouvez-vous utiliser pour détecter un circuit dans un graphe orienté ?	☐ parcours en largeur ☐ parcours en profondeur ☐ algorithme de Dijkstra ☐ algorithme de Bellman
4. Quel algorithme faut-il privilégier pour calculer un plus court chemin dans un graphe orienté *sans circuit* dans lequel tous les coûts sont *positifs* ?	☐ algorithme de Bellman ☐ algorithme de Prim ☐ algorithme de Bellman-Ford ☐ algorithme de Dijkstra
5. Quelle est la complexité d'un algorithme du type "diviser pour régner" qui divise un problème de taille n en deux sous-problèmes de taille $n/2$ et dont l'opération de fusion est en $\Theta(n^2)$?	☐ $\Theta(n)$ ☐ $\Theta(n \log n)$ ☐ $\Theta(n^2)$ *suite sur la page suivante...*

Questions	Réponses
	☐ $\Theta(n^3)$
6. Exécuter l'algorithme de Dijkstra sur le graphe G_1 de la figure 1, afin de trouver une arborescence des plus courts chemins de racine 1. Dans quel ordre les sommets sont-ils examinés ? Vous *représenterez* également sur le graphe G_1 *l'arborescence des plus courts chemins* retournée par l'algorithme de Dijkstra (vous pouvez surligner les arcs sélectionnés).	☐ (1,2,3,4,5,6,7) ☐ (1,4,2,5,3,6,7) ☐ (1,2,4,5,3,6,7) ☐ (1,4,2,5,6,3,7)

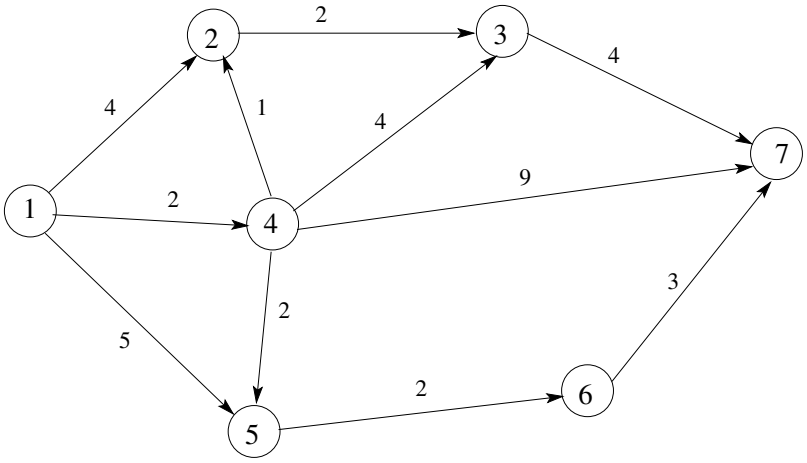


FIGURE 1 – Graphe G_1.

Exercice 2 (7 points)

Vous souhaitez concevoir une projection de photos pour vos ami(e)s. Supposons que vous ayez déjà classé n photos $\{1, \ldots, n\}$ dans l'ordre dans lequel vous souhaitez les projeter. Vous avez maintenant la possibilité de mettre une ou deux photos par affichage lors de la projection. Mettre deux photos côté à côte peut en effet faire un joli effet, par exemple si les photos ont le même sujet. Pour $i \in \{1, \ldots, n-1\}$, vous notez $v_i \in \{0, 1\}$ la valeur de l'effet attendu si les photos i et $i+1$ sont projetées côte à côte : si $v_i = 1$, placer les photos i et $i+1$ côte à côte provoque un joli effet, et si $v_i = 0$, placer les photos i et $i+1$ côte à côte ne provoque pas de joli effet. Votre but est de maximiser le nombre de jolis effets pendant la projection de photos.

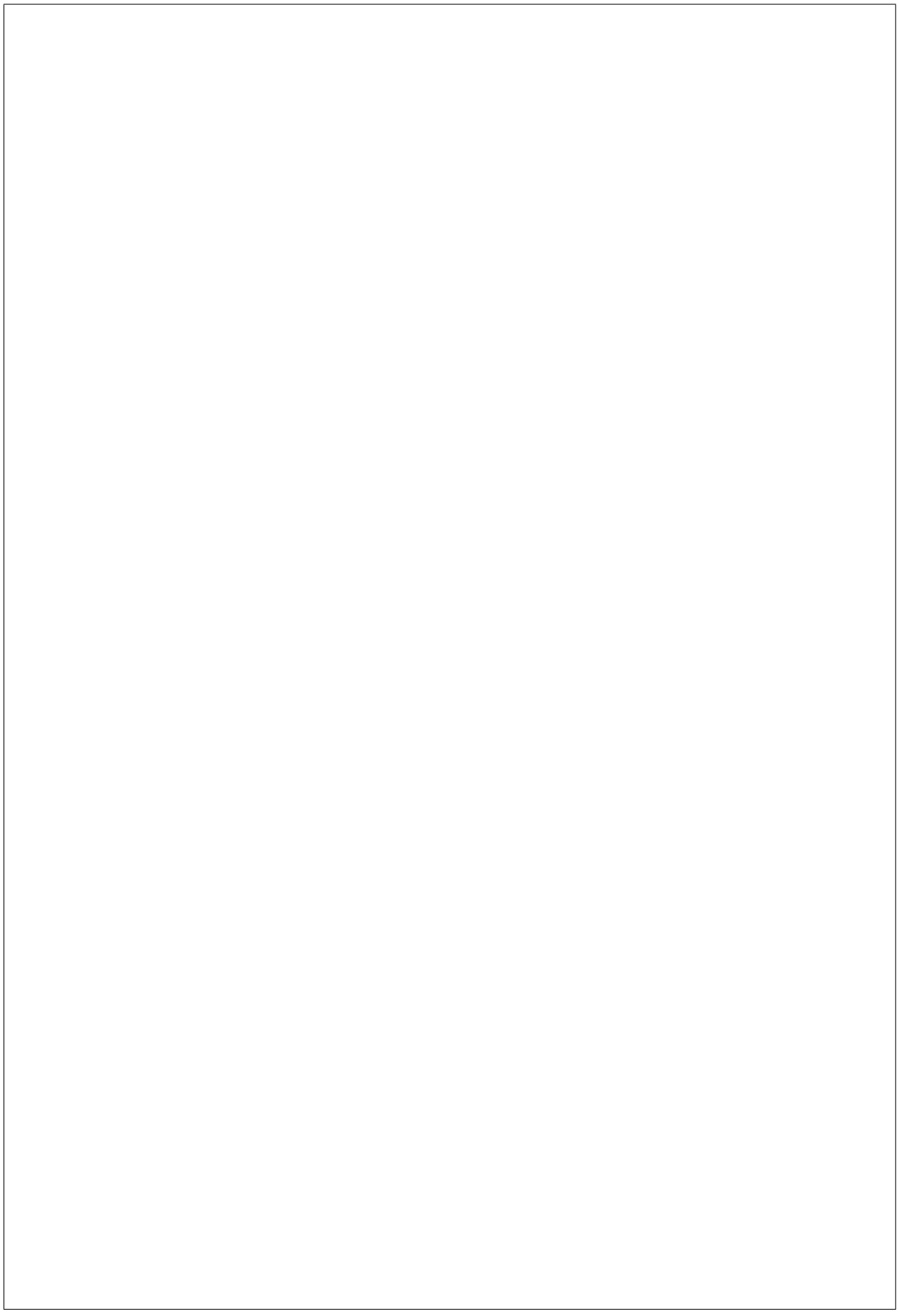
Votre petite sœur vous propose de considérer les photos dans l'ordre de 1 à n et de mettre côte à côte deux photos dès qu'elles créent un joli effet. Par exemple, dans le tableau V de la figure 2, vous placerez les photos 2 et 3 côte à côte puis les photos 5 et 6 côte à côte, suivies des photos 7 et 8 côte à côte.

0	1	1	0	1	1	1

FIGURE 2 – Exemple de tableau V pour une projection de 8 photos, avec $V[i] = v_i$. Placer les photos 1 et 2 côte à côte ne crée pas de joli effet ($V[1] = 0$), alors que placer les photos 2 et 3 côte à côte crée un joli effet ($V[2] = 1$).

Question 1 (2.5/7) — Prouver que cet algorithme maximise bien le nombre de jolis effets. Vous veillerez à être rigoureux/rigoureuse dans votre preuve.

Remarque : si lors de la projection, l'affichage des photos obtenu avec l'algorithme induit k jolis effets, on pourra noter $E = \{e_1, \ldots, e_k\}$ l'ensemble des photos placées avec une autre photo et à gauche, lors de la projection (i.e. $e_i = k$ si les photos k et $k+1$ sont projetées côte à côte). Dans l'exemple plus haut, on aurait eu $E = \{2, 5, 7\}$ car les jolis effets concernent les photos 2 et 3, présentées ensemble, puis les photos 5 et 6, puis les photos 7 et 8.



On suppose maintenant que tous les jolis effets ne se valent pas : certains sont plus intéressants que d'autres. Pour $i \in \{1, \ldots, n-1\}$, on notera $v_i \in \mathbb{N}^+$ la valeur de l'effet attendu si les photos i et $i+1$ sont projetées côte à côte. Si $v(i) = 0$, il semble équivalent de passer les photos i et $i+1$ chacune sur une page, ou de les placer côte à côte. Si $v_i > 0$, alors placer les photos côte à côte provoque un joli effet, et plus $v(i)$ est important, plus l'effet est joli.

Vous souhaitez concevoir un algorithme de programmation dynamique permettant de déterminer quelles paires de photos présenter côte à côte (en respectant l'ordre initial choisi) de manière à maximiser la somme des effets attendus : $\sum_{i \in \{1,\ldots,n-1\} | i \text{ et } i+1 \text{ sont placées côte à côte}} v_i$.

Dans un premier temps, nous chercherons uniquement à calculer la somme maximale des effets attendus – on notera cette valeur OPT –, et pas à savoir quelles photos placer côte à côte.

Pour tout $i \in \{2, \ldots, n\}$, notons $S(i)$ la valeur maximale des effets que l'on peut obtenir en considérant les photos de 1 à i uniquement (ces photos étant placées dans l'ordre $1, 2, \ldots, i$). On fixera $S(1) = 0$ (il n'y a pas d'effet si l'on considère une photo uniquement).

Question 2 (0.5/7) — Exprimer OPT en fonction des valeurs $S(i)$.

Question 3 (1.5/7) —

a) Exprimer, en fonction d'une ou plusieurs valeur(s) $S(k)$, avec $k < i$ la valeur de $S(i)$ si l'on sait que la photo i est projetée seule.

b) Exprimer, en fonction d'une ou plusieurs valeur(s) $S(k)$, avec $k < i$ la valeur de $S(i)$ si l'on sait que la photo i est projetée côte à côte avec la photo $i-1$.

c) Donner une formule de récurrence permettant de calculer la valeur $S(i)$, pour tout $i \in \{3, \ldots, n\}$.

Question 4 (1.5/7) — Écrire un algorithme de programmation dynamique calculant OPT. Quelle est la complexité de votre algorithme ?

Question 5 (1/7) — Modifier votre algorithme pour qu'il affiche les pages côte à côte. Par exemple, il affichera "(2,3) (6,7)" si dans la solution optimale les photos 2 et 3 puis 6 et 7 sont affichées côte à côte. Vous pouvez pour cela ajouter une fonction supplémentaire.

Exercice 3 (11 points)

Étant donné un graphe $G = (S, A)$, un *approximant* (V, R) est un couple de parties disjointes de A tel qu'il existe un arbre couvrant de coût minimum OPT qui contient toutes les arêtes de V et aucune arête de R.

Ainsi, chaque arête de $e \in A$ peut soit appartenir à V, soit à R, soit ni à V ni à R. Nous dirons par la suite que les arêtes de V sont vertes, et celles de R sont rouges. Les arêtes de A qui ne sont ni dans V ni dans R seront dites libres, et considérées comme blanches. Un approximant est donc un sous-ensemble d'arêtes de A colorées en vert et en rouge, tel qu'il existe un arbre couvrant de coût minimum contenant l'ensemble des arêtes vertes et aucune arête rouge.

Exemple : sur la figure ci-dessous, (V, R) est un approximant avec $V = \{\{A, B\}, \{A, C\}\}$ (arêtes en gras plein) et $R = \{\{B, C\}\}$ (arête en pointillé). En effet, il existe un arbre couvrant de coût minimum $(\{\{A, B\}, \{A, C\}\}, \{C, D\})$ contenant les arêtes de V et pas celle de R.

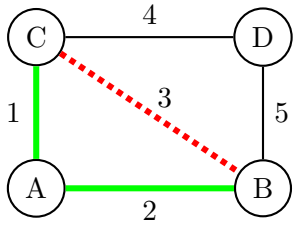


FIGURE 3 – Exemple d'approximant.

Question 1 (3/11) — Exécuter l'algorithme de Kruskal sur le graphe G_2 représenté dans la figure 4. Vous indiquerez, à chaque itération de votre algorithme, un approximant correspondant à l'exécution de l'algorithme. Vous surlignerez également l'arbre couvrant de coût minimum obtenu à la fin de l'exécution de l'algorithme.

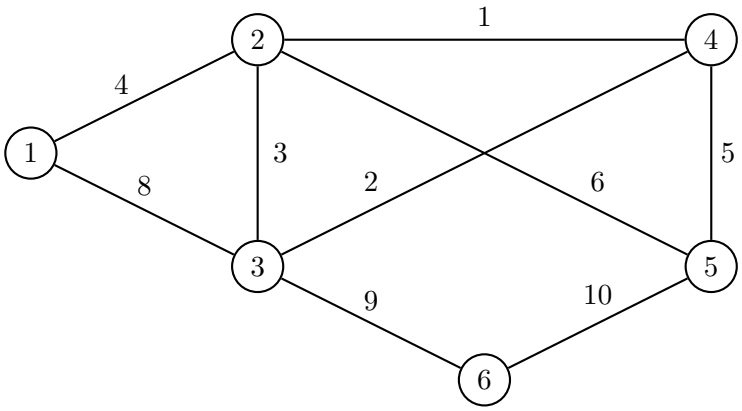


FIGURE 4 – Graphe G_2.

Les questions suivantes concernent *un graphe non orienté, connexe, et pondéré* $G = (S, A)$ *quelconque* (et non le graphe G_2 de la question précédente).

Question 2 (1/11) — Soit (V, R) un approximant de G.

a) Montrer que le graphe $G' = (S, V)$ est une forêt couvrante de G.

b) Que peut-on dire si $|V| = n - 1$? Justifiez votre réponse.

Deux règles permettent d'ajouter une arête (verte ou rouge) à un approximant : *la règle des cocycles*[1] et *la règle des cycles*. Ces règles permettent de créer un algorithme générique de construction d'arbre couvrant de coût minimum : en partant de l'approximant $(\emptyset, \emptyset)$ on ajoute peu à peu une arête rouge ou une arête verte jusqu'à obtenir un arbre couvrant de coût minimum. Nous verrons que les algorithmes de Prim et de Kruskal sont des cas particuliers de cet algorithme générique.

Dans cet exercice, vous prouverez la règle des cocycles (questions 3 à 6), et admettrez la règle des cycles. En utilisant ces règles, vous prouverez dans les questions 7 et 8 que les algorithmes de Prim et de Kruskal retournent bien des arbres couvrants de coût minimum.

La règle des cocycles.

Étant donné un approximant (V, R), la règle des cocycles va nous permettre d'ajouter une arête verte à V en colorant en vert l'arête blanche la moins chère du cocycle :

> **Règle des cocycles :** Soit $G = (S, A)$ un graphe connexe non orienté et pondéré. Soit (V, R) un approximant de G, et soit ω un cocycle dont chaque arête est soit rouge soit blanche. Soit $e \in A$ une arête blanche de ω de coût minimum. $(V \cup \{e\}, R)$ est un approximant.

Les questions suivantes concernent un graphe $G = (S, A)$ connexe non orienté et pondéré quelconque. Soit (V, R) un approximant de G et soit OPT un arbre couvrant de coût minimum de G tel que OPT contient les arêtes de V mais pas celles de R. Supposons qu'il existe dans G un cocycle ω formé d'arêtes rouges (i.e. de R) ou blanches (i.e. de $A \setminus \{V \cup R\}$).

Question 3 (0.5/11) — Pourquoi peut-on être sûr qu'il existe nécessairement une arête blanche dans ω ?

Soit $e = \{x, y\}$ une arête blanche de ω de coût minimum.

Question 4 (0.5/11) — La taille d'un approximant (V, R) est $|V \cup R|$: le nombre d'arêtes colorées en vert ou en rouge. Donner un approximant de taille $|V| + |R| + 1$ si $e \in OPT$. On ne demande pas de justification.

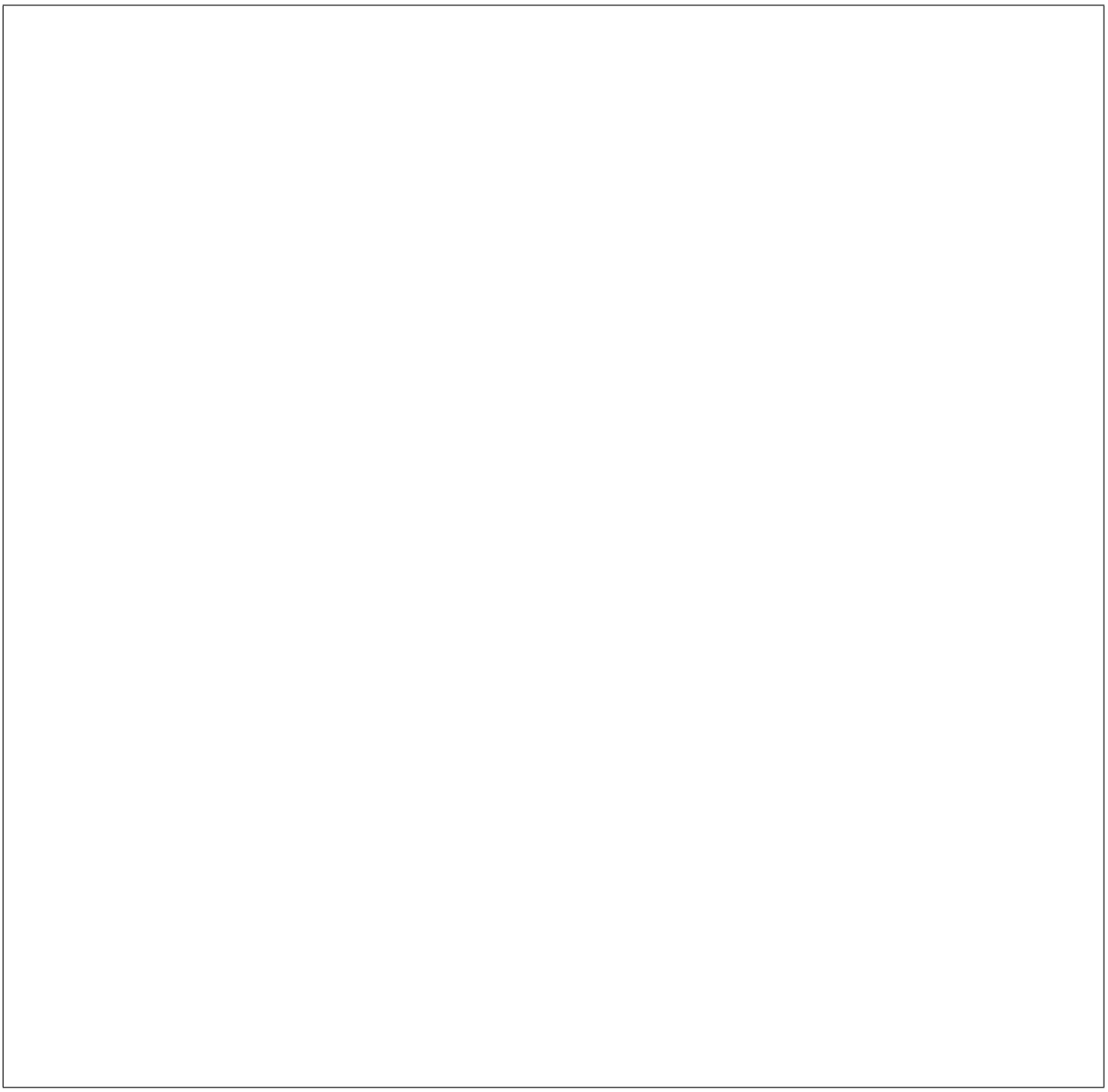
1. rappel : Soit $S' \subset S$ un sous ensemble des sommets d'un graphe $G = (S, A)$. Le *cocycle* $\omega(S')$ associé à S' est l'ensemble des arêtes qui ont une extrémité dans S' et une extrémité qui n'est pas dans S'.

Supposons maintenant que $e \notin OPT$.

Question 5 (1/11) — Soit γ la chaîne qui relie x à y, les deux extrémités de e, dans l'arbre couvrant OPT. Montrer que γ contient nécessairement une arête f qui est blanche et telle que $c(f) \geq c(e)$.

Question 6 (2/11) — Donner un approximant de taille $|V|+|R|+1$ si $e \notin OPT$. Justifiez votre réponse.

Indice : on pourra considérer le graphe partiel $X = (OPT \setminus \{f\}) \cup \{e\}$.



Découvrons maintenant la règle des cycles, qui permet d'ajouter une arête rouge à un approximant.

La règle des cycles.

La règle des cycles consiste à dire qu'étant donné un approximant et un cycle ne contenant pas d'arête rouge, on peut colorer en rouge l'arête blanche la plus chère du cycle et garder un approximant. Plus formellement :

> **Règle des cycles :** Soit $G = (S, A)$ un graphe connexe non orienté et pondéré. Soit (V, R) un approximant de G, et soit $\mathcal{C}$ un cycle dont chaque arête est soit verte soit blanche. Soit $e \in A$ une arête blanche de $\mathcal{C}$ de coût maximum. $(V, R \cup \{e\})$ est un approximant.

Par manque de temps, on admettra cette règle sans la prouver. Vous pourrez donc supposer par la suite que les règles des cycles et des cocycles ont toutes les deux été prouvées.

Un algorithme générique.

Les algorithmes de Prim et de Kruskal, comme d'autres, sont des algorithmes gloutons qui reposent sur l'application de la règle des cycles et des cocycles. En effet, tout algorithme glouton qui procède par itération, en colorant à chaque étape une arête en rouge ou en vert, en appliquant la règle des cycles ou des cocycles, va, au bout d'un certain nombre d'itérations, retourner un arbre couvrant de coût minimum.

Question 7 (1/11) — Rappeler en quelques mots le principe de l'algorithme de Prim en expliquant en quoi cet algorithme utilise la règle des cocycles et/ou la règle des cycles. En déduire la validité de l'algorithme de Prim.

Question 8 (2/11) — Rappeler en quelques mots le principe de l'algorithme de Kruskal en expliquant en quoi cet algorithme utilise la règle des cocycles et/ou la règle des cycles. En déduire la validité de l'algorithme de Kruskal.