N° d'anonymat:

Année Universitaire 2023-2024
Session 1 de printemps

Parcours/Étape: L3 Info./MI/CMI **Code UE:** 4TIN602U
Épreuve: Techniques Algorithmiques et Programmation
Date: 15/04/2024 **Heure:** 11h30 **Durée:** 1h30
Documents: une seule feuille A4 recto-verso autorisée.
Épreuve de M. Cyril Gavoille

université de BORDEAUX

Collège Sciences et Technologie

RÉPONDRE DIRECTEMENT SUR LE SUJET
QUI EST À RENDRE DANS LA FEUILLE DOUBLE D'EXAMEN

Question de personnes

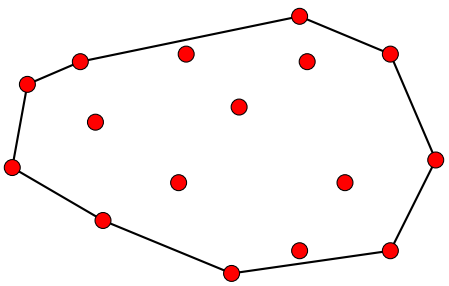
Question 1. *Parmi les personnes suivantes qui était responsable du Cours, et qui était responsable de votre TD/TP ? [Cochez la ou les cases correspondantes.]*

COURS	*TD/TP*
☐ *Pierre Bonnet*	☐ *Pierre Bonnet*
☐ *Charles Brazier*	☐ *Charles Brazier*
☐ *Cyril Gavoille*	☐ *Cyril Gavoille*
☐ *Vincent Pennelle*	☐ *Vincent Pennelle*
☐ *Zoé Varin*	☐ *Zoé Varin*
☐ *Marc Zeitoun*	☐ *Marc Zeitoun*
☐ *[autre]*	☐ *[autre]*

Solution. **[1 pts |∑1.0]** Pour la colonne 1 : Cyril Gavoille. Pas de point attribué pour la colonne 2, toutes les réponses étant valables.

Voyageur de commerce

Soit V un ensemble de points du plan, et d la distance euclidienne. L'*enveloppe convexe* de V est la plus petite courbe fermée dont l'intérieur contient tous les points de V.



On peut montrer que les segments de droite connectant deux points de l'enveloppe convexe sont toujours contenus à l'intérieur de l'enveloppe *convexe*. On dira que V est convexe si tous les points

de V appartiennent à son enveloppe convexe. C'est par exemple le cas des points situés sur un cercle ou sur un polygone régulier. Ce n'est pas le cas des points de la figure ci-dessus.

Question 2. *Montrez que si V est convexe alors la longueur de la tournée optimale du* Voyageur de commerce *pour (V, d) est la longueur de l'enveloppe convexe, c'est-à-dire que la tournée optimale est obtenue en visitant les points dans l'ordre de l'enveloppe convexe.*

Solution. **[3 pts |∑4.0]** L'enveloppe convexe est la plus petite courbe dont l'intérieur contient tous les points. Si tous les points sont sur cette courbe, c'est-à-dire si V est convexe, alors la tournée obtenue à partir de l'enveloppe convexe est la plus courte possible visitant tous les points.

Autre argument : si on ne suit pas l'enveloppe convexe, alors la tournée aurait un croisement (à l'intérieur de l'enveloppe) et la tournée ne serait pas optimale comme vu en TD.

Plus formellement, supposons l'ordre $v_0, \ldots, v_{n-1}$ des points selon l'enveloppe convexe avec au moins 4 points (sinon c'est trivialement vrai). Si la tournée ne suit pas cet ordre, c'est qu'elle contient un segment $v_i - v_j$ entre deux points du cercle avec au moins un point de part et d'autre de ce segment, disons v_k et v_t. Le chemin C issu de la tournée connectant v_k à v_t (disons v_k visité avant v_t) ne peut repasser ni par v_i ni par v_j. Il y a donc un segment $v_{i'} - v_{j'}$ de C qui croise $v_i - v_j$, ce qui montre que la tournée n'est pas optimale : contradiction.

On rappelle le principe de l'algorithme ApproxMST vu en cours.

Algorithme ApproxMST(V, d)

Entrée : Une instance (V, d) du Voyageur de commerce.
Sortie : Une tournée, c'est-à-dire un ordre sur les points de V.

1. Calculer un arbre couvrant de poids minimum T sur le graphe complet défini par V et les arêtes valuées par d.
2. La tournée est définie par l'ordre de première visite des sommets selon un DFS de T, c'est-à-dire un parcours en profondeur d'abord.

Question 3. *Rappelez ce qu'est un arbre couvrant de poids minimum.*

Solution. **[3 pts |∑7.0]** Il s'agit d'un arbre dont les arêtes couvrent tous les points de V et tel que la somme des poids des arêtes est la plus petite possible.

Question 4. *Donnez, en fonction de $|V|$, le nombre d'arêtes du graphe complet à l'étape 1.*

Solution. **[3 pts |∑10.0]** C'est $\binom{|V|}{2} = |V|(|V|-1)/2$.

On a vu dans le cours que l'algorithme ApproxMST est une 2-approximation.

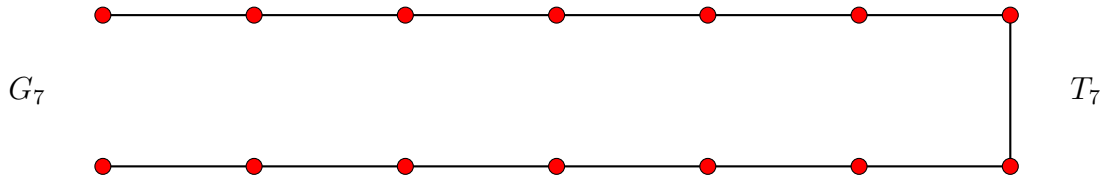
Question 5. *Qu'est-ce qu'une 2-approximation pour le* Voyageur de commerce *?*

Solution. **[3 pts |∑13.0]** C'est un algorithme polynomial qui produit une tournée de longueur au plus deux fois plus que l'optimale.

L'objectif est de montrer que le facteur d'approximation d'ApproxMST peut se rapprocher arbitrairement près de 2, même dans le cas euclidien. Pour cela, on considère G_n comme l'ensemble des points d'une grille $2 \times n$, où deux points consécutifs sur l'enveloppe convexe sont à distance 1. Soit T_n l'arbre, en fait le chemin, constitué de l'enveloppe convexe de G_n moins un segment vertical. Voir l'exemple ci-dessous pour $n = 7$.

Question 6. *Quelle est la longueur de la tournée optimale pour (G_n, d) ? Justifiez.*

Solution. **[3 pts |∑16.0]** La tournée optimale est le longueur $2n$. En effet, on ne peut pas faire moins car G_n possède $2n$ points et la distance entre deux points quelconques est au moins 1. Toute



tournée passant par tous les points est donc de longueur au moins $2n$. On remarque aussi que G_n est convexe.

Question 7. *Montrez que T_n est bien un arbre couvrant de poids minimum et donnez son poids.*

SOLUTION. **[3 pts |∑19.0]** T_n est un arbre couvrant G_n. Donc, il possède $|E(T_n)| = |V(G_n)| - 1 = 2n - 1$ arêtes. Chaque arête du bord a un poids de 1. Son poids est donc $2n - 1$.

Question 8. *Montrez qu'il existe un sommet de T_n où tout DFS depuis ce sommet produit une tournée de longueur optimale.*

SOLUTION. **[3 pts |∑22.0] Remarque importante.** Il exite plusieurs parcours DFS pour un même arbre (par exemple en choisissant une racine plutôt qu'une autre, ou en partant dans une branche plutôt qu'une autre). En particulier, l'étape 2 de l'algorithme ApproxMST peut produire potentiellement plusieurs tournées plusqu'elle choisit "un DFS de T" parmi les possibles pour T.

Prendre une des feuilles de T_n. Il n'y a qu'un seul parcours possible (car T_n est un chemin) qui donne une longueur de $2n$, ce qui est optimale d'après la question 6 précédente.

Question 9. *Montrez qu'il existe un DFS de T_7 où la tournée résultante est de longueur strictement comprise entre 24 et 26. Précisez le premier sommet et le parcours, par exemple en les dessinant. [Aide : Vous pourrez utiliser le fait que $x < \sqrt{x^2+1} < x+1$ pour tout réel $x > 0$.]*

SOLUTION. **[3 pts |∑25.0]** Je présente ici la solution du cas général. Il faut prendre comme sommet de départ un coin de G_n qui ne soit pas une feuille de T_n. Le DFS doit partir d'abord vers la branche la plus courte, celle horizontale avec $n-1$ arêtes. La longueur de la tournée résultante (qui fait un beau croisillon) est alors $\ell(n) = 2(n-1) + 2\sqrt{(n-1)^2+1}$. Il faut connaître la formule de la distance euclidienne.

Pour $n = 7$, cela donne donc $\ell(7) = 2 \cdot 6 + 2\sqrt{6^2+1} \in]24, 26[$, ou encore $24 < \ell(7) < 26$.

Question 10. *Quel est dans ce cas le facteur d'approximation ? (dans les conditions de la question 9, c'est-à-dire avec le DFS particulier de T_7). Donnez un encadrement.*

SOLUTION. **[3 pts |∑28.0]** D'après la question 6, il suffit de diviser $\ell(n)$ par $2n = 14$ pour avoir le facteur d'approximation d'ApproxMST. Ce facteur est donc dans l'intervalle $]24/2 \cdot 7, 26/2 \cdot 7[=]12/7, 13/7[$, ou encore $1+]5/7, 6/7[$.

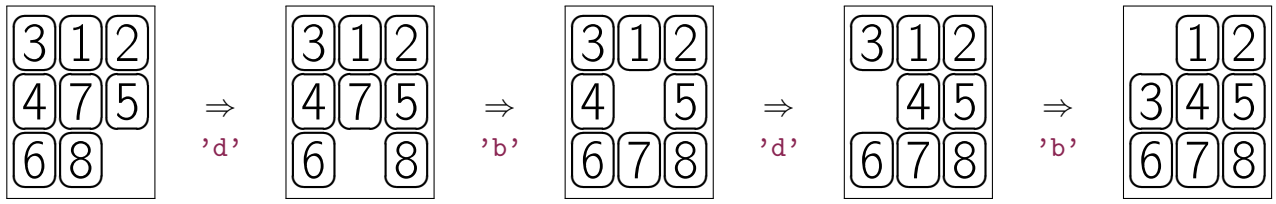
Question 11. *En généralisant les questions 9 et 10 à tout entier $n > 7$, montrez que le facteur d'approximation d'ApproxMST pour (G_n, d) vaut $2 - O(1/n)$.*

SOLUTION. **[3 pts |∑31.0]** On a vu que la longueur de la tournée est $\ell(n) = 2(n-1) + 2\sqrt{(n-1)^2+1} \in]4n-4, 4n-2[$. Le facteur d'approximation est donc dans l'intervalle $](4n-4)/2n, (4n-2)/2n[=]2 - 2/n, 2 - 1/n[= 2-]2/n, 1/n[$ ce qui tend vers $2 - O(1/n)$.

Jeu de Taquin

On considère le Taquin d'ordre n, un jeu à un joueur composé de $n^2 - 1$ carreaux numérotés de 1 à $n^2 - 1$ qui glissent dans un cadre $n \times n$, avec une case vide donc. Il consiste à remettre dans l'ordre tous les carreaux à partir d'une configuration initiale quelconque. Un carreau ne peut se déplacer que s'il est voisin de la case vide (4-voisinage). Une fois effectué, le déplacement crée une nouvelle configuration avec une autre case vide.

Ci-dessous, un exemple de configuration de Taquin d'ordre $n = 3$ (figure de gauche), où quatre déplacements permettent de gagner, c'est-à-dire d'atteindre la configuration gagnante (figure de droite).



On représentera une configuration S par un tableau 2D où `S[i][j]` est un entier unique de $[0, n^2[$, la case vide étant codée par 0. La case `S[i][j]` correspond au carreau situé sur la i-ème ligne et j-ème colonne, `S[0][0]` étant située en haut à gauche.

Question 12. *Indiquez, en fonction de n, le nombre de configurations possibles. Justifiez.*

SOLUTION. **[3 pts |∑34.0]** Il y a $(n^2)!$ configurations possibles, car il s'agit d'une permutation des valeurs de $\{0, \ldots, n^2 - 1\}$.

Suivant la configuration de départ, il n'est pas toujours possible d'atteindre la configuration gagnante. C'est par exemple le cas si comme configuration de départ on choisit la configuration gagnante dans laquelle on a échangé les deux derniers carreaux.

On cherche à déterminer la séquence de déplacements la plus courte permettant de gagner, lorsque c'est possible. On considère d'abord une approche exhaustive.

Une séquence de k déplacements va être codée par un mot de k lettres sur l'alphabet `{'d','g','h','b'}`. On le représente par un tableau `char D[k]`, où `D[i]` codera le i-ème déplacement d'un carreau, voisin de la case vide,

- vers la droite ($\rightarrow$), si `D[i]='d'`
- vers la gauche ($\leftarrow$), si `D[i]='g'`
- vers le haut ($\uparrow$), si `D[i]='h'`
- vers le bas ($\downarrow$), si `D[i]='b'`.

Le mot correspondant aux quatre déplacements de l'exemple précédent est `"dbdb"`. Certains mots ne correspondent pas à une séquence valide, comme `"bbbbbb"` pour $n = 3$, ou encore `"d"` ou `"b"` pour une configuration gagnante, car la case vide en tête n'a pas de carreau adjacent qui puisse venir vers la droite ou du bas.

Question 13. *Écrivez en `C` une fonction `bool valid(int **S,int n,char *D,int k)` qui renvoie `true` si et seulement si la séquence des k déplacements codée dans D est valide pour la configuration de départ S (d'ordre n). [Aide : Assurez-vous que la case vide ne sorte pas du cadre.]*

SOLUTION. **[5 pts |∑39.0]**

```
bool valid(int **S,int n,char *D,int k){
      int i,j,i0,j0;

      // cherche la position i0,j0 de la case vide
      for(i=0; i<n; i++)
        for(j=0; j<n; j++)
          if(S[i][j]==0) i0=i, j0=j, i=j=n;

      // vérifie chaque déplacement de la case vide
      for(i=0; i<k; i++)
        switch(D[i]){ // sort si déplacement invalide
          case 'd': if( --i0 <  0 ) return false; break;
          case 'g': if( ++i0 == n ) return false; break;
          case 'h': if( --j0 <  0 ) return false; break;
          case 'b': if( ++j0 == n ) return false; break;
        }

      return true; // tous les déplacements sont valides
    }
```

Question 14. *Quelle est, en fonction de n et k, la complexité de votre implémentation. Justifiez.*

SOLUTION. **[3 pts $|\sum 42.0|$]** C'est $O(n^2+k) : O(n^2)$ à cause de la recherche de la case vide, puis $O(k)$ pour la vérifications des k déplacements.

On admettra que la séquence la plus courte permettant de gagner au Taquin d'ordre n, si elle existe, contient au plus $n \log_2 n$ déplacements. Et ceci quelle que soit la configuration de départ S.

On considère le code suivant permettant de trouver la séquence minimum de déplacements pour gagner depuis la configuration S, si c'est possible (renvoie `NULL` sinon). Pour cela on suppose déjà codées les fonctions :

- `win(S,n,D,k)` indiquant si une séquence de déplacements D permet d'atteindre la configuration gagnante depuis la configuration S.
- `next_move(D,k)` permettant d'énumérer toutes les séquences de k déplacements, en passant à la prochaine (renvoie `false` seulement si elles ont toutes été passées en revue)

```
char *taquin_brute_force(int **S,int n){
  int N = n*log2(n); // nombre de déplacements maximum
  char *D = malloc(N*sizeof(*D)); // séquence de déplacements à trouver

  for(int k=0; k<N; k++){ // k = nombre de déplacements
    for(int i=0; i<k; i++) D[i]='d'; // initialisation de D
    do{ if ( valid(D,k) && win(S,n,D,k) ) return D; // solution trouvée
        } while (next_move(D,k)); // prochaine séquence de k déplacements
  }

  free(D);
  return NULL; // il n'y a pas de solution
}
```

Question 15. *En supposant que chacune des fonctions `valid()`, `win()` et `next_move()`, sont de complexité $O(n^c)$ pour une certaine constante $c > 0$, donnez en fonction de n et c la complexité pour la fonction `taquin_brute_force(S,n)`. Justifiez. [Aide : Vous pourrez utiliser le fait que $x^{\log_2 y} = y^{\log_2 x}$*

pour $x, y > 0$.]

SOLUTION. **[3 pts |∑45.0]** La boucle principale, `for(int k=0;...)` comprend deux instructions qui sont deux boucles.

- La complexité de la boucle `for(int i=0;...)`, pour l'initialisation de `D`, est $O(k)$.
- La complexité de la boucle `do{...}while(...)` est $4^k \times O(n^c)$, car, pour une longueur k donnée, il y a 4^k séquences possibles, et chacune des trois fonctions du `do{...}` a une complexité $O(n^c)$ par hypothèse.

La complexité de $C(n)$ de `taquin_brute_force()` est donc majoré par :

$$C(n) \;=\; \sum_{k=1}^{N} \left(O(k) + 4^k \times O(n^c)\right) = \left(\sum_{k=1}^{N} O(k)\right) + \left(O(n^c) \times \sum_{k=1}^{N} 4^k\right) \;.$$

Clairement, la première somme est plus petite que la deuxième. Donc, à un facteur deux près, $C(n)$ est majorée par :

$$\begin{aligned} C(n) &\leqslant O(n^c) \times \sum_{k=1}^{N} 4^k = O(n^c) \times O(4^N) \\ &\leqslant O(n^c) \times O(4^{n \log_2 n}) = O(n^c) \times O(n^{2n}) = O(n^{2n+c}) \;. \end{aligned}$$

On considère une autre approche basée sur l'algorithme A*. Pour cela, on définit le graphe des configurations, noté $\mathcal{C}_n$, dont les sommets sont toutes les configurations possibles du Taquin d'ordre n. Deux sommets X et Y sont connectés si un déplacement permet de passer de la configuration X à la configuration Y. Ce graphe est non valué, chaque arête a un poids de 1.

Dans la suite on notera T la configuration gagnante.

Question 16. *Expliquez les points faibles d'une approche qui utiliserait l'algorithme Dijkstra pour le calcul d'un plus court chemin d'une configuration de départ S à la configuration T dans le graphe $\mathcal{C}_n$. [Aide : Approximez $x! \approx x^x$.]*

SOLUTION. **[2 pts |∑47.0]** Il y a un problème d'espace mémoire. En effet, Dijkstra nécessite l'intialisation d'une table de taille au moins le nombre de sommets du graphe, ici $(n^2)! \approx n^{2n^2}$, un nombre gigantesque. La complexité en temps sera elle aussi plus grande que pour A*.

Note. En fait, le majorant annoncé sur k de $N = n \log_2 n$ n'est pas correct. Il est de $5n^3$ (et au moins de $n^3 - O(n)$) d'après un article de Parberry de 1995, cf. doi :10.1016/0020-0190(95)00134-X. Cela n'impacte aucune des questions, puisqu'on pouvait y répondre correctement en supposant que $N = n \log_2 n$ comme proposé.

On définit pour $\mathcal{C}_n$ l'heuristique $h(X, T)$ comme le nombre de carreaux de la configuration X qui ne sont pas positionnés à la même place dans T, la case vide n'étant pas prise en compte.

On a par exemple $h(X_1, T) = 4$, si X_1 est la configuration la plus à gauche de l'exemple de départ. En effet, seuls les quatre carreaux 3,4,7,8 sont mal positionnés par rapport à la configuration gagnante. D'ailleurs, on peut vérifier que pour les cinq configurations $X_1, X_2, X_3, X_4, X_5 = T$ de l'exemple, on a $h(X_i, T) = 5 - i$.

Question 17. *Démontrez que h est monotone. [Aide : Montrez qu'un déplacement de X à Y ne peut pas "trop" diminuer $h(X, T)$.]*

SOLUTION. **[3 pts |∑50.0]** Il faut montrer que lors d'un déplacement de $X \to Y$, $h(X, T) \leqslant 1 + h(Y, T)$. Cela revient à démontrez que $h(Y, T) \geqslant h(X, T) - 1$. C'est en effet le cas, car lors d'un déplacement de X à Y, un seul carreau se déplace. Donc au mieux ce carreau de X passe de "mal" à "bien" positionné, une diminution d'au plus une unité sur $h(X, T)$. Ce qui fait que $h(Y, T) \geqslant h(X, T) - 1$.

Un étudiant propose une heuristique alternative h_2 pour accélérer la recherche du plus court chemin vers T. Il pose $h_2(X, T)$ comme la somme, sur tous les carreaux y compris la case vide, de la distance entre sa position dans X et sa position dans T. Ici la distance est calculée selon le 4-voisinage.

Par exemple, pour la configuration X_1, on a $h_2(X_1, T) = [1 + 0 + 0] + [1 + 1 + 0] + [0 + 1 + 4] = 8$. Bien sûr $h_2(T, T) = 0$, comme précédemment.

Question 18. *Discutez si l'utilisation de h_2 pour A* vous semble une bonne idée. Argumentez pour ou contre.*

SOLUTION. **[2 pts** $|\sum 52.0$**]** Le problème est que h_2 n'est pas monotone, et donc ne pourra garantir que la solution trouvée sera de plus court chemin. En effet, chaque déplacement peut aboutir à la diminution de deux unités l'heuristique (à cause de la case vide !). Il aurait fallut ne pas tenir compte des déplacements de la case vide.

FIN.