TD1.4 – Baseball multicolore

Objectifs

À la fin de cette séance, vous devriez être capable de :
— décrire un algorithme en langue naturelle ;
— décomposer l'exécution d'un algorithme en actions indépendamment de l'instance du problème ;
— prouver votre algorithme (correction, terminaison) ou trouver un contre-exemple à sa validité ;

Exercice 1 :

1. Cherchez à résoudre le problème.
2. Cherchez une méthode permettant de résoudre toutes les instances du problème (un algorithme).
3. Exprimez cet algorithme le plus précisément possible en langue naturelle.
4. Vérifiez si votre algorithme est suffisamment clair :
   — Donnez une feuille avec votre algorithme rédigé à un autre groupe (appelons-le A).
   — Logiquement, un autre groupe (appelons-le B) doit aussi vous donner sa feuille à ce moment du TD.
   — En groupe, lisez attentivement l'algorithme que le groupe B vous a communiqué, essayez de le comprendre, de l'exécuter, voire de déterminer s'il vous semble correct ou non.
   — Désignez un rapporteur dans votre groupe, qui va aller réexpliquer leur *propre algorithme* au groupe B.
   — À ce moment, un rapporteur du groupe A devrait venir vous réexpliquer votre algorithme. Écoutez-le attentivement sans l'interrompre.
   — Que pensez-vous de ses explications ? Quels points de votre algorithme a-t-il compris de travers ? Comment pouvez-vous modifier votre rédaction pour supprimer ces incompréhensions ?

   Jeu de rôle : faites exécuter l'algorithme par un camarade sans regarder l'instance du problème.
5. — Prouvez que votre algorithme est correct et qu'il se termine. Allez en 6.
   ou
   — Trouvez un contrexemple montrant que votre algorithme est incorrect ou ne se termine pas. Allez en 2.
6. Calculez l'ordre de grandeur du coût de votre algorithme en nombre de déplacements en fonction du nombre de bases.

Correction de l'exercice 1

Si le problème n'est pas très compliqué à résoudre sur une configuration donnée, trouver une méthode générale est plus difficile. L'intérêt de ce TD réside donc dans la recherche d'une méthode et surtout dans sa rédaction.

En guise de correction, on propose ici plusieurs « algorithmes » et on discute brièvement le principe qui a guidé leur conception, leurs défauts et leurs mérites respectifs.

Algorithme tournant Dans n'importe quelle configuration du problème, on a un choix à faire entre 4 coups distincts (2 joueurs sur chacune des 2 bases adjacentes au trou).

À première vue, il n'est pas facile de choisir le « bon » coup parmi ces quatre, ni d'exprimer simplement comment on fait ce choix. Pour réduire cet espace de choix, on décide de ne tourner que dans un seul sens (horaire par exemple). Ainsi, un coup se résume à décider lequel des 2 joueurs on fait avancer ; et il n'est alors pas très compliqué de voir qu'il est plus productif de déplacer le joueur qui a la plus grande distance à parcourir avant d'arriver à sa base (si la distance est la même, c'est que les deux joueurs ont la même couleur ; les deux coups sont alors équivalents).

Tant que tous les joueurs ne sont pas rentrés à leur base, on continue les déplacements. Cette méthode est simple à expliquer, et dans la plupart des cas, elle résoud le problème relativement rapidement. Malheureusement, il existe des états de départs pour lesquels elle ne permet pas d'arriver à la solution. Pour s'en convaincre, il suffit de prendre le jeu dans son état résolu, et

d'intervertir deux joueurs. On observe alors qu'on repasse périodiquement par cet état initial sans atteindre la solution, et donc qu'on ne s'arrêtera jamais.

Cette méthode ne mérite donc que le nom de « semi-algorithme », ou d'algorithme *partiellement correct*. Plus le nombre de bases augmente, plus les configurations non-terminantes seront d'ailleurs nombreuses.

Algorithme en ligne Pour ne pas tomber dans l'écueil précédent, il faut donc s'assurer que l'algorithme « progresse ». Pour cela, une méthode consiste à imaginer une barrière sur le cercle, infranchissable par les joueurs. On a donc à nouveau restreint l'espace des coups possibles, mais cette fois en transformant le cercle en ligne.

Par commodité, on s'arrange pour que la base n'ayant qu'un seul joueur soit à l'extrémité droite de cette ligne ; on va alors s'occuper des bases les unes après les autres, de gauche à droite sur cette ligne. Pour rapprocher un joueur de sa base, il suffit de créer un trou à gauche de ce joueur. Et pour cela, il suffit de déplacer les joueurs des autres couleurs (peu importe lesquels) pour amener le trou à gauche du joueur à déplacer (en fait pour cette étape il est plus pratique de considérer qu'on « déplace le trou »).

On répète l'opération jusqu'à ce que les deux joueurs de la base la plus à gauche soient revenus à leur base, et on n'y touche plus. On peut maintenant ignorer la première base, qui est déjà rentrée, et on recommence avec les bases restantes.

La correction de l'algorithme est assez claire (lorsqu'on s'arrête, c'est qu'on a rempli toutes les bases correctement). La terminaison est un peu plus subtile mais on y arrive en reprenant les étapes :

— mettre le trou à gauche d'un joueur demande au maximum $n-1$ déplacements ;
— mettre un joueur sur sa base demande au maximum $n-1$ « déplacements du trou » et $n-1$ déplacements dudit joueur ;
— remplir une base demande de mettre 2 joueurs sur cette base ;
— résoudre le problème demande de remplir correctement $n-1$ bases (et la base la plus à droite sera automatiquement correcte).

Ce raisonnement nous donne d'ailleurs une première estimation (très) grossière du coût de l'algorithme : moins de $(n-1)\times(n-1)\times 2\times(n-1)$ déplacements.

On peut en fait affiner ce premier calcul, car la plupart du temps on n'aura à déplacer le trou que de 2 bases (d'un côté du pion à l'autre). En comptant en même temps les déplacements du trou et ceux du pion qu'on cherche à déplacer, on obtient un nombre de coups pour placer un pion sur sa base de l'ordre de $4n$, et donc au total un algorithme en $\mathcal{O}(n^2)$.

Algorithme en ligne : variante shaker Cet algorithme est un raffinement du précédent.

Tout d'abord, on voit qu'on peut placer la base à un seul joueur à n'importe quel endroit de la ligne, et remplir les bases en commençant par les extrémités. Expérimentalement, il est alors un peu meilleur de placer la base à un joueur en position centrale.

Mais surtout, il va s'agir de faire en sorte, autant que possible, que *chaque déplacement* nous rapproche de la solution. On se concentre donc sur les « déplacements du trou » et on choisit avec soin le joueur à déplacer pour cela.

On peut alors procéder de la façon suivante :

— on place initialement le trou tout à gauche ;
— on alterne ensuite entre une phase où le trou se déplace de l'extrémité gauche à l'extrémité droite, et inversement, jusqu'à avoir résolu le problème ;
— pour déplacer le trou de gauche à droite, on l'intervertit avec le joueur place à sa droite et dont la base est la plus à gauche possible ;
— et inversement lorsqu'il s'agit de déplacer le trou de droite à gauche.

Ainsi, la plupart des coups consistent à rapprocher un joueur de sa base.

De plus, lorsque les bases des extrémités sont correctement remplies, il n'est bien entendu plus nécessaire d'y passer, on peut donc progressivement réduire l'espace au sein duquel le trou fait ses allers-retours.

On peut montrer que cet algorithme est en $\mathcal{O}(n^2)$; en revanche, pour rendre la constante devant n^2 intéressante il faut éliminer des déplacements parasites qui se produisent dans certaines situations.

Décomposition en sous-problèmes On a déjà appliqué en partie cette démarche de décomposition pour l'algorithme linéaire : pour remplir une base il suffit de savoir rapprocher un joueur, et pour rapprocher un joueur il suffit de savoir déplacer le trou.

On peut se poser la question en ces termes : existe-t-il une opération de base telle que, si je sais l'effectuer, alors je suis sûr de savoir résoudre le problème ? On trouve pour cela deux bons candidats :

1. Échanger 2 joueurs voisins sans modifier les autres.
2. Échanger un joueur donné avec le trou sans modifier les autres joueurs.

La première opération est relativement simple à effectuer (4 déplacements si le trou est déjà adjacent à l'un de nos 2 joueurs, auxquels il suffit alors d'ajouter $2 \times \frac{n}{2}$ déplacements pour rapprocher le trou puis le remettre à sa place). Puisqu'on sait échanger 2 joueurs voisins, on peut implémenter un tri à bulles.

La seconde opération est un peu plus complexe (ce qui est normal car elle nous permet d'effectuer un déplacement global et non plus local) mais elle reste en $\mathcal{O}(n)$ déplacements :

— sur tout le chemin entre le joueur à déplacer et le trou, on chaque base contient deux joueurs que l'on nomme A et B ;
— on déplace chaque joueur A vers le trou ;
— on rapproche le joueur qu'on voulait déplacer de sa destination en l'échangeant avec un B, puis en ramenant un A à sa place ;
— en répétant cette série de 3 déplacements, le joueur arrive à sa destination, les A reviennent à leur place initiale, et les B sont décalés ;
— enfin on remet les B à leur place, ce qui a aussi pour effet d'emmener le trou à l'emplacement initial du joueur qu'on voulait déplacer.

On applique cette procédure pour placer à chaque fois un joueur de la bonne couleur sur la base qui contient le trou ; dans le cas où le trou arriverait sur la base à un seul joueur et où ce joueur y serait déjà, il suffit de mettre dans le trou n'importe quel joueur mal placé. Ainsi, on résoud le problème en effectuant au pire $3n$ appels à une procédure linéaire (le lecteur intéressé essayera de comprendre pourquoi $3n$).