Les algorithmes seront écrits en code CAML et seront accompagnés de commentaires et d'explications qui permettront de les comprendre aisément.

Problème 1 - Anatomie d'un disque dur

Dans un disque dur, il y a souvent plusieurs têtes de lecture (ou écriture, mais on dira toujours lecture seulement) : dans ce problème, on supposera qu'il y en a deux. Un disque dur est divisé en secteurs numérotés de 1 à N.

Au départ, les deux têtes de lecture sont au repos (en position 0). Lorsque un disque reçoit des demandes de lecture ou écriture, il n'obéit pas tout de suite : il enregistre les demandes dans un tampon de longueur n et il répond aux demandes lorsque le tampon est plein. À ce moment-là, le système de contrôle du disque dur va déplacer les têtes de lecture pour lire ou écrire : il agit de sorte que les têtes se déplacent le moins possible, pour ne pas perdre de temps dans les déplacements, tout en respectant l'ordre dans lequel les demandes sont arrivées. Il doit donc décider quelle tête de lecture va répondre à chaque demande. Le coût d'un déplacement entre les secteurs numérotées p et q est supposé être proportionnel à $|p-q|$: quitte à multiplier tous les résultats par une constante, on suppose dans la suite que le coût est égal à $|p-q|$.

Par exemple, si on suppose que $n=4$ et que les demandes concernent des lectures ou écritures sur les secteurs $8, 1, 5, 3$, alors

- s'il décide de confier toutes les demandes à la tête numéro 1, celle-ci va se déplacer sur le secteur 8 (coût du déplacement 8), puis sur le secteur 1 (coût du déplacement 7), puis sur le secteur 5 (coût 4), enfin le secteur 3 (coût 2) : le total des coûts de déplacements est $8+7+4+2=21$;
- une meilleure solution est donné par exemple en confiant les demandes concernant les secteurs 8 et 5 à la tête 1 et les deux autres à la tête 2 : le coût des déplacements de la tête 1 est alors $8+3=11$ et celui de la tête 2 est $1+2=3$, soit un coût total de 14.

L'objectif est d'écrire un algorithme efficace qui calcule la répartition des demandes entre les deux têtes de lecture, de sorte que la somme des déplacements des deux têtes soient minimales.

On représente une suite de demandes par un tableau d de longueur $n+1$: la première case du tableau contient 0 (la position initiale des têtes) et chaque case numéro i ($i \geqslant 1$) du tableau contient un entier compris entre 1 et N. En ajoutant 0 en tête du tableau, on a ainsi la suite des positions qui doivent être atteintes par l'ensemble des deux têtes de lecture : la demande numéro 0 consiste donc à rester en position initiale.

Pour $k \in \{1, \dots, n\}$, pour traiter les demandes numérotées de 0 à k, l'une des deux têtes va traiter la dernière demande. On note alors i le numéro de la **dernière** demande traitée par l'autre tête : i est un entier de $\{0, \dots, k-1\}$ (si $i=0$, alors une tête n'a pas bougé et l'autre tête a traité toutes les demandes). On note alors $c(k,i)$ le coût minimal des déplacements des têtes pour traiter les demandes numérotées de 0 à k, sachant que la demande numéro i a été traitée par une tête et les suivantes numérotées de $i+1$ à k par l'autre tête. On appelle $R(k,i)$ le problème qui consiste à traiter les demandes numérotées de 1 à k avec l'une des deux têtes qui effectue sa dernière action en traitant la demande numéro i. Une solution optimale à un tel problème est une solution dont le déplacement total vaut $c(k,i)$.

Partie 1 - Étude du problème

Question 1) Que vaut $c(1,0)$ en fonction des éléments de d ? Plus généralement, que vaut $c(k,0)$?

Question 2) Justifiez que le coût minimal des déplacements traitant toutes les demandes est le nombre S égal au minimum des $c(n,i)$ pour i variant de 0 à $n-1$.

Question 3) Soit $k \in \{2, \dots, n\}$ et $i \in \{0, \dots, k-2\}$. Que dire des têtes de lecture qui ont traitées les demandes numérotées $k-1$ et k dans une répartition optimale du problème $R(k,i)$? Montrez alors que $c(k,i) = c(k-1,i) + \big|d[k]-d[k-1]\big|$.

Question 4) Soit $k \in \{2, \dots, n\}$ et $i = k-1$. Que dire des têtes de lecture qui ont traitées les demandes numérotées $k-1$ et k dans une répartition optimale du problème $R(k,k-1)$? Montrez alors que $c(k,k-1) = \min\limits_{0 \leqslant j \leqslant k-2} \big(c(k-1,j) + \big|d[k]-d[j]\big|\big)$.

Question 5) Quel style de programmation vous paraît le plus adapté à la résolution de ce problème ? Justifiez votre choix.

Question 6) Exemple : on donne $d = (0;8;1;2;6;3)$.

Dans un tableau à 5 lignes indicées par k variant de 1 à 5 et à 5 colonnes indicées par i variant de 0 à $k-1$, donnez les valeurs des nombres $c(k,i)$.

Donnez la valeur du coût minimal des déplacements des têtes. En observant plus attentivement ce tableau, comment peut-on en déduire une répartition des demandes entre les deux têtes ?

Partie 2 - Calcul de S et plus si affinités

Question 1) Écrivez une fonction `minimum f n` de type `(int -> int) -> int -> int`, qui calcule le minimum des entiers `f 0`, `f 1`, ... , `f n` (on suppose que le paramètre `n` est un entier naturel et le paramètre `f` une fonction qui calcule un entier à partir d'un entier).

Question 2) Écrivez une fonction `cout_deplacements d` de type `int vect -> int`, qui calcule le nombre S (objectif : moins de 15 lignes normales).

Question 3) Quel est la complexité de la fonction précédente ?

Question 4) Proposez des modifications pour calculer en plus la répartition des demandes sur chaque tête : vous choisirez la représentation de ces répartitions.

Devoir non surveillé 5 - Corrigé

Problème 1

Partie 1

Question 1) S'il n'y a qu'une demande à traiter, elle est attribuée à une tête, l'autre restant immobile : le coût total est donc $|d[1] - d[0]| = d[1]$.

Plus généralement, sur la colonne 0, on trouve le coût des déplacements tous effectués par une même tête, car l'autre est restée en position 0 : la tête a donc effectué les déplacements $|d[1] - d[0]| + |d[2] - d[1]| + \ldots + |d[k-1] + d[k]|$.

Question 2) La tête b a traité la dernière demande dans une répartition optimale. L'autre tête a a traité sa dernière demande avant (éventuellement aucune) : soit i l'indice de la dernière demande traitée par a ($i = 0$ si a est restée immobile). Le coût des déplacements des deux têtes est donc le nombre $c(n, i)$, et c'est bien le plus petit, sinon il y aurait une autre répartition qui aurait un coût plus petit, ce qui est contradictoire.

Question 3) Soit $k \in \{2, \ldots, n\}$ et $i \in \{0, \ldots, k-2\}$. On considère une solution optimale du problème $R(k, i)$. La dernière demande numéro k a été traitée par la tête b et la dernière demande traitée par a est celle numéro i, avec $i < k - 1$, donc toutes celles qui suivent ont été traitées par la même tête, la tête b par conséquent : en particulier, la tête b a traité la demande $k - 1$ avant la demande k. Son dernier déplacement vaut donc $|d[k] - d[k-1]|$.

Les demandes précédentes forment nécessairement une solution optimale du problème $R(k-1, i)$: si ce n'est pas le cas, en ajoutant le dernier déplacement de la tête b à une autre solution optimale du problème $R(k-1, i)$, on a une meilleure solution au problème $R(k, i)$, ce qui est contradictoire.

Donc $c(k, i) = c(k-1, i) + \big|d[k] - d[k-1]\big|$.

Question 4) Soit $k \in \{2, \ldots, n\}$ et $i = k - 1$. On considère une solution optimale du problème $R(k, k-1)$. Par définition, une tête (disons la b) a traité la dernière demande numéro k et le rang de la dernière demande traitée par l'autre tête est $k - 1$, donc les demandes numérotées $k - 1$ et k sont traitées par des têtes différentes.

L'avant-dernière demande traitée par b est en position $j \leqslant k - 2$, la répartition des demandes numérotées de 0 à $k - 1$ est alors une solution au problème $R(k-1, j)$ et elle est optimale (même raisonnement : par l'absurde). Son coût est donc $c(k-1, j)$ et donc le coût de la solution optimale du problème $R(k, k-1)$ est $c(k-1, j) + \big|d[k] - d[j]\big|$, puisque avant de traiter la dernière demande, la tête b est en position $d[j]$.

Bien sûr, ce nombre $c(k-1, j) + \big|d[k] - d[j]\big|$ est bien le plus petit des nombres de ce type (pour j variant de 0 à $k - 2$), par le même raisonnement par l'absurde.

Question 5) On a une formulation récursive d'un problème d'optimisation, qui vérifie le principe de sous-optimalité de Bellmann (c'est l'objet des question précédentes, justement), avec un grand nombre d'appels récursifs identiques. La programmation dynamique est toute indiquée pour résoudre ce genre de problème.

Question 6) k en ligne, i en colonne :

	0	1	2	3	4
1	8				
2	15	9			
3	16	10	15		
4	20	14	19	12	
5	23	17	22	15	13

Sur la ligne 1, le seul nombre est le coût du premier déplacement : la première demande est attribuée à l'une ou l'autre des têtes, disons la première.

Ensuite, on passe à la ligne 2 : le minimum sur cette ligne est à droite (c'est $c(2, 1)$), donc il a été calculé par la relation de la question 4, ce qui signifie qu'on a changé de tête de lecture. On attribue donc la demande numéro 2 à la tête 2.

Puis sur la ligne 3, le minimum est à la même position que celui de la ligne précédente (c'est $c(3, 1)$), donc il a été calculé par la relation de la question 3, donc la tête de lecture précédente a été conservée, donc on attribue la demande numéro 3 à la tête 2.

Sur la ligne suivante, le minimum est de nouveau au bout à droite : la demande numéro 4 est attribuée à l'autre tête, donc on revient à la tête 1.

Et sur la dernière ligne, c'est pareil, on change de nouveau de tête : la demande numéro 5 est attribuée à la tête 2.

Finalement, la tête 1 va traiter les demandes 1 et 4, la tête 2 va traiter les demandes 2, 3 et 5.

Partie 2

Question 1)

```
let rec minimum f n =
  if n = 0 then f 0
  else min (minimum f (n-1)) (f n);;
```

Question 2)

```
let cout_deplacements d =
  let n = vect_length d -1  in
  let cout = make_matrix (n+1) (n) 0 in
  let rec dep k i =
    if cout.(k).(i) <> 0 then cout.(k).(i)
    else let r =
      if k = 1 then d.(1)
      else if i < k - 1 then dep (k-1) i + abs(d.(k-1) - d.(k))
      else let f x = dep (k-1) x + abs (d.(x) - d.(k)) in
           minimum f (k-2)
    in cout.(k).(i) <- r; r
  in
  minimum (dep n) (n-1);;
```

Question 3) La fonction remplit les cases d'une matrice de taille $(n+1) \times n$: pour chaque case, quelques opérations élémentaires sont utilisées, sauf pour la dernière de chaque ligne, qui nécessite de parcourir la ligne précédente pour en calculer le minimum.

Pour remplir la ligne 1, le coût est de 1.

Pour remplir la ligne 2, le coût est de $5 + 1 \times 5 = 10$: le premier 5 est le calcul direct par la relation de la question 3, partie 1, le second est le parcours de la ligne précédente avec le même coût unitaire.

Pour remplir la ligne 3, le coût est de $5 + 5 + 2 \times 5 = 20$: idem

Pour remplir la ligne 4, le coût est de $5 + 5 + 5 + 3 \times 5 = 30$

Etc.

Pour remplir la ligne n, le coût est de $5 + 5 + \ldots + 5 + (n-1) \times 5 = 10(n-1)$.

Le coût total du remplissage du tableau est donc de $1 + 10 + 20 + \ldots + 10(n-1) = O(n^2)$.

Comme les appels récursifs ne sont faits qu'une seule fois (les appels identiques suivants sont en fait des lectures de case d'une matrice, donc de coût $O(1)$), le coût total de la fonction est donc du même ordre de grandeur que celui du remplissage du tableau, donc finalement la complexité est en $O(n^2)$.

Question 4) La fonction minimum2 calcule non seulement le minimum, mais aussi le rang donnant le minimum.

La fonction chemin prend en paramètre le tableau cout des coûts successifs calculés, le numéro k de la ligne sur laquelle on travaille, le rang du minimum dans la ligne de rang k+1 et les deux listes contenant les répartitions calculées avec les demandes numérotées de $k+1$ à n.

```
let rec minimum2 f n =
  if n = 0 then f 0, 0
  else let r = minimum2 f (n-1) in
  if fst r < f n then r else (f n, n);;

let rec chemin cout k j l1 l2 =
  if k = 0 then (l1,l2)
  else let f x = cout.(k).(x) in
  let (_, j') = minimum2 f (k - 1) in
  if j = j' then chemin cout (k-1) j (k :: l1) l2
  else chemin cout (k-1) j' (k :: l2) l1 ;;

let depl2 d =
  let n = vect_length d - 1  in
  let cout = make_matrix (n+1) (n) 0 in
  let rec dep k i =
    if cout.(k).(i) <> 0 then cout.(k).(i)
```

```
    else let r =
      if k = 1 then d.(1)
      else if i < k - 1 then dep (k-1) i + abs(d.(k-1) - d.(k))
      else let f x = dep (k-1) x + abs (d.(x) - d.(k)) in
        minimum f (k-2)
    in cout.(k).(i) <- r; r
  in
  let m, j = minimum2 (dep n) (n-1) in
  m, chemin cout (n-1) j [n] [];;
```