ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Section d'Informatique et de Systèmes de Communication

Corrigé de la série 9 23 Novembre 2009

1. Application de QuickSort *et* HeapSort

a) **Suite originale:** Le pivot est 48. On a un pointeur p qui commence a droite de la liste, qu'on déplace vers la gauche en s'arrêtant dès qu'on arrive à un élément plus petit que le pivot. De même on a un pointeur q qui commence à gauche et qu'on déplace vers la droite en s'arrêtant dès qu'il arrive à un élément plus grand que le pivot.

Ainsi p va d'abord s'arrêter à 10, q à 84. On échange ces deux éléments. Puis on continue, p s'arrête à 1, q à 58, on échange c es deux éléments etc. Lorsque p et q sont arrêteś et se sont croisés on a fini la première étape. On inverse le pivot avec l'élément vers lequel pointe q. Et on divise notre suite en deux sous-suites: la première consiste en tous les éléments à gauche du pivot (ils sont tous plus petits que lui par construction), et la deuxième consiste en tous les éléments à droite du pivot (il sont tous plus grands par construction). On applique ensuite QuickSort récursivement à ces deux sous-suites.

Dans notre cas les éléments suivants sont donc échangés:

$$10-84, 1-58, 10-81, 4-49, 10-91, 33-89$$

Quand p et q se sont croisés, q pointe vers 63. On échange donc 48 (le pivot) et 63. Et on coupe la suite en deux sous-suites: les éléments à gauche et à droite de 48

Sous-suite de gauche: La suite à gauche de 48. C'est la suite suivante:

$$23, 29, 10, 15, 1, 19, 10, 17, 48, 15, 36, 4, 10, 26, 33, 22$$

On prend 22 comme pivot, et comme ci-dessus on échange les éléments suivants:

$$10-23, 4-29, 15-48$$

Quand p et q s'arrêtent et se sont croisés, q pointe vers 48. On échange donc 48 avec 22 (le pivot), et on a deux sous suites:

$$10, 4, 10, 15, 1, 19, 10, 17, 15 \qquad \text{(à gauche de 22)}$$

$$36, 29, 23, 26, 33, 48 \qquad \text{(à droite de 22)}$$

On va donc appliquer QuickSort à ces deux sous suites (avec comme pivots 15 et 48). etc...

Sous-suite de droite: La suite à droite de 48:

$$57, 89, 91, 50, 56, 85, 49, 81, 63, 58, 72, 84, 63$$

Le pivot est 63, on échange les éléments suivants:

$$58 - 89, 49 - 91$$

Quand p et q se sont croisés q pointe vers 85. On échange donc 85 avec le pivot et se retrouve avec deux sous-suites:

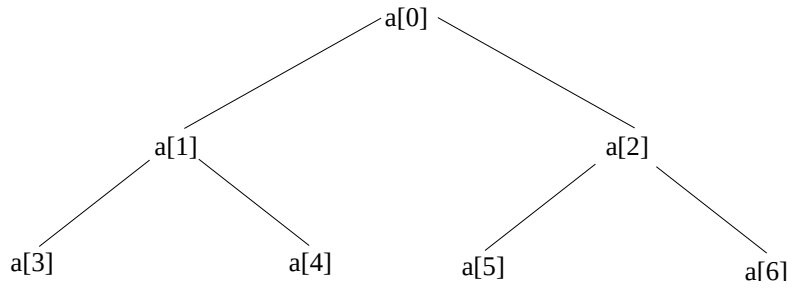
$$57, 58, 49, 50, 56 \qquad \text{(à gauche de 63)}$$

$$91, 81, 63, 89, 72, 84, 85 \qquad \text{(à droite de 63)}$$

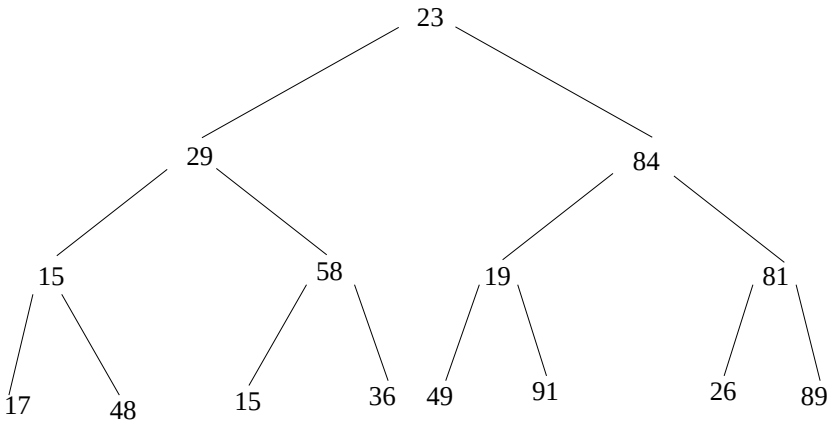
On va donc trier chacune de ces sous suites avec QUICKSORT, les pivots seront donc 56 et 85.

...etc...

b) On rappelle qu'on représente une suite $a[0], \ldots, a[N-1]$ par un arbre en prenant $a[0]$ comme racine, puis $a[1]$ et $a[2]$ pour ses deux fils, et en général $a[2i+1]$ et $a[2i+2]$ pour les deux fils de $a[i]$, comme suit:



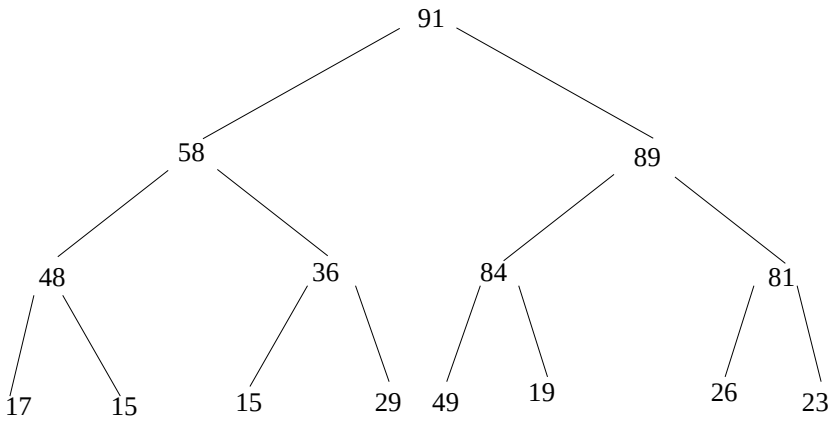
Si on représente notre suite par un arbre, elle est dans l'état initial suivant:



Il faut d'abord transformer cette suite en heap. On utilise BottomUpHeapCreate. On va donc faire des opérations de sift down (sur les éléments $a[6], a[5], \ldots, a[0]$, donc $81, 19, \ldots, 23$ puisque les autres sont déjà "en bas" de l'arbre):

- Sift down de 81: échanger 81 et 89
- Sift down de 19: échanger 19 et 91
- Sift down de 58: rien
- Sift down de 15: échanger 15 et 48
- Sift down de 84: échanger 84 et 91
- Sift down de 29: échanger 29 et 58, échanger 29 et 36
- Sift down de 23: échanger 23 et 91, échanger 23 et 89, échanger 23 et 81

Ainsi on se retrouve avec le heap suivant:



on voit que c'est bien un heap puisque chaque sommet est plus grand que ses deux fils.

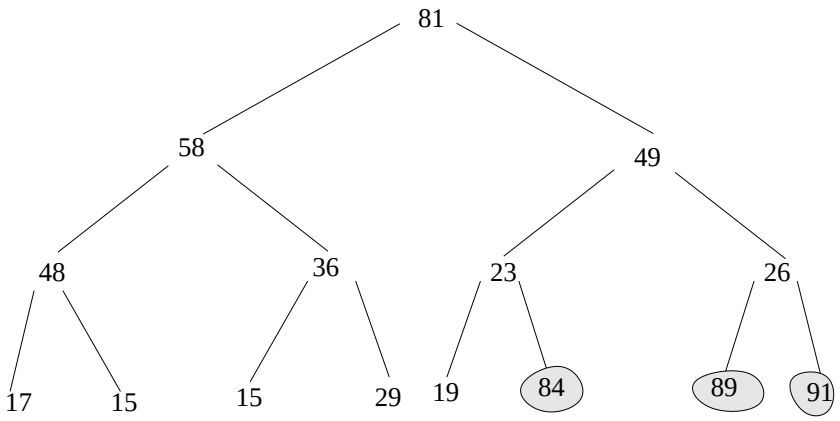
Nous pouvons donc procéder avec l'algorithme heap sort:

- On échange 23 et 91. On ne touche plus à 91, puisqu'on sait que c'est le plus grand élément il est donc à sa place à la fin de la liste. On fait ensuite un sift down de 23 (mais sans toucher à 91): échanger 23 et 89, échanger 23 et 84, échanger 23 et 49.

- On échange 26 et 89. De même on ne touche plus à 89 ou 91 puisque ce sont les deux plus grands éléments de la liste, ils sont donc à leur place. On fait un sift down de 26: échanger 26 et 84, échanger 26 et 81.

- On échange 19 et 84. On fait un sift down de 19 sans toucher à 84, 89 et 91: échanger 19 et 81, échanger 19 et 49, échanger 19 et 23.

Notre liste est à présent la suivante:



Les éléments 84, 89 et 91 sont les plus grands éléments de la suite, ils sont donc a leur place on ne va plus les toucher. On continue de la même façon:

• On échange 19 et 81. On fait un sift down de 19 sans toucher à 81, 84, 89 et 91: échanger 19 et 58, échanger 19 et 48

• etc...

2. Dégénérescence de QuickSort

a) Si la suite est déjà triée, le pivot sera à chaque fois le plus grand élément. La suite sera donc coupée en une suite vide (à droite) et une suite consistant en tous les éléments sauf le pivot (à gauche). Il faudra donc faire $N-1$ étapes qui nécessiteront $1, \ldots, N-1$ comparaisons, donc au total $\Omega(N^2)$.

b) Il s'agit à nouveau de construire des suites de sorte que le pivot soit fréquemment un élément très grand (ou très petit) de la suite. Cette stratégie empêche qu'on puisse arranger la suite de sorte que le pivot soit le plus grand élément de la suite, mais c'est toujours possible que ça soit le deuxième plus grand élément.

Supposons que les clés sont les nombres $0, 1, \ldots, N-1$, et que N est pair. Alors l'arrangement

$$0, *, \cdots, *, (N-1), *, \cdots, *, a, (N-2),$$

(où le "$(N-1)$" se trouve à la position $N/2$, et les "$*$" dénotent n'importe quelle clé), fait que l'algorithme choisit $(N-2)$ comme pivot, puisque c'est la médiane de $(0, N-1, N-2)$. Après le premier pas de partitionnement de QuickSort on aura

$$\underline{0, *, \cdots, *, a, *, \cdots, *}, (N-2), (N-1),$$

où les $*$ n'ont pas changé de position. (Remarquons que QuickSort effectue $N-2$ comparaisons pour arriver à cette étape.)

On veut que le comportement pathologique ci-dessus se répète: pour y arriver, on peut disposer les éléments dans la sous-suite soulignée ci-dessus de manière analogue, i.e., positionner $(N-3)$ au milieu et $(N-4)$ tout à droite. Donc l'arrangement

$$0, *, \cdots, *, N-3, N-1, *, \cdots, *, N-4, *, N-2$$

fait que les deux premières itérations exhiberont un comportement pathologique.

En répétant la construction, on peut s'arranger pour que les premières $\lfloor N/4 \rfloor$ itérations seront de ce genre:

$$0, *, \cdots, N-\frac{2N}{4}-1, \cdots, N-3, N-1, \cdots, *, N-\frac{2N}{4}, *, N-\frac{2N}{4}+2, *, \cdots, N-4, *, N-2.$$

Pour effectuer chacune de ces itérations, $\geq N/2$ comparaisons sont nécessaires, donc au total $\geq N/4 \cdot N/2 = \Omega(N^2)$ comparaisons.

Remarque: Comme il n'était pas précisé que les éléments de la suite doivent être *distincts*, on peut trouver une solution bien plus simple: Si tous les éléments sont égaux ($a[0] = \ldots = a[N-1]$) il faudra $\Omega(N^2)$ comparaisons.

En effet, à chaque passage le pointeur p ne s'arrêtera que lorsqu'il arrivera au début de la suite (la position ℓ dans l'algorithme du cours), alors que q ne s'arrêtera qu'à la position $r-1$. Ainsi la sous-suite de droite sera toujours vide. Il faudra donc faire un passage de QuickSort sur une suite de taille N, puis $N-1$, puis $N-2$, etc... Comme chaque passage nécessite $\Omega(N)$ comparaisons, il faudra $\Omega(N^2)$ comparaisons au total.

3. Nombre moyen d'échanges avec Selection Sort

a) Voici le tableau d'échanges s'il n'y a que trois éléments:

permuation	nombre d'échanges
012	0
021	1
102	1
120	2
201	2
210	1

En effet, 012 est déjà ordonné donc aucun échange n'est nécessaire. Pour 021 il suffit d'échanger 1 et 2, pour 102 on échange 0 et 1. Pour 120 on échange d'abord 0 et 1, puis 1 et 2, donc deux échanges sont nécessaires. Et ainsi de suite.

Si toutes ces permutations ont la même probabilité, le nombre moyen d'échanges sera

$$\frac{0+1+1+2+2+1}{6} = \frac{7}{6}.$$

b) (i) Il y a $k!$ permutations de $0, \ldots, k-1$ au total (il y a k éléments). Nous voulons compter dans combien d'entre elles 0 est à la bonne place. Pour construire une permutation dans laquelle 0 est à la bonne position, on le place d'abord en premier, puis on peut placer les $k-1$ éléments qui restent de n'importe quelle façon (donc selon n'importe laquelle des $(k-1)!$ permutations possibles). Ainsi la probabilité que 0 est à la bonne place est

$$\frac{(k-1)!}{k!} = \frac{(k-1)\cdots 2 \cdot 1}{k \cdot (k-1)\cdots 2 \cdot 1} = \frac{1}{k}.$$

(ii) Comme expliqué dans la question, SELECTIONSORT fait $N-1$ itérations pour i allant de 0 à $N-2$. Pendant l'itération i on fait un échange si et seulement si le $i^{\text{ème}}$ élément n'est pas à la bonne place.

Pour la première itération ce sera la cas avec probabilité $1-\frac{1}{N}$ (d'après la question (i), où ici on a N éléments donc $k=N$).

Le nombre moyen d'échanges durant la première itération est donc

$$1\cdot\left(1-\frac{1}{N}\right)+0\cdot\frac{1}{N}=1-\frac{1}{N}=\frac{N-1}{N}.$$

De même, au début de la $i^{\text{ème}}$ itération on sait que les éléments $0,\ldots,i-1$ sont tous à la bonne position. Il reste donc $N-i$ éléments disposés de façon aléatoire dans la liste. Il faudra faire un échange si et seulement si l'élément i n'est pas à la bonne position, ce qui arrive avec probabilité $1-\frac{1}{N-i}$ (question (i) avec $N-i$ éléments). Comme ci-dessus, le nombre moyen d'échanges pendant la $i^{\text{ème}}$ itération est donc

$$\frac{N-i-1}{N-i}.$$

Finalement, le nombre moyen d'échanges au total est égal à la somme des nombres moyens d'échanges à chaque itération, donc

$$\frac{N-1}{N}+\frac{N-2}{N-1}+\ldots+\frac{2}{3}+\frac{1}{2}.$$

4. Traverser des graphes

a) 1,12,4,6,9,10,11,3,2,5,8,7 (cette solution n'est pas unique, puisque l'algorithme n'est pas deterministe).

b) 1,12,5,8,4,7,6,2,9,10,11,3 (cette solution n'est pas unique).

• On commence par mettre 1 dans la queue Q.
• On ajoute tous ses voisins non-marqués à Q: 12, 5, 8 (qu'on marque aussi dans cet ordre). On a donc $Q=(1,12,5,8)$.
• On fait dequeue, on a donc: $Q=(12,5,8)$.
• On prend le premier élément de Q: 12. On ajoute tous ses voisins non marqués: 4 (on marque aussi 4). On a donc $Q=(12,5,8,4)$.
• On fait dequeue, on a donc: $Q=(5,8,4)$.
• On prend le premier élément de Q: 5. On ajoute tous ses voisins non marqués à Q: il n'y en a pas.
• On fait dequeue, donc $Q=(8,4)$.
• On prend le premier élément de Q: 8. On ajoute tous ses voisins non marqués à Q: 7. On a donc $Q=(8,4,7)$.
• Dequeue, donc $Q=(4,7)$.
• On prend le premier élément de Q: 4. On ajoute tous ses voisins non marqués à Q: $6,2$. On a donc $Q=(4,7,6,2)$.

- Dequeue, donc $Q = (7, 6, 2)$.
- On ajoute tous les voisins non marqués de 7 à Q: $9, 10$. On a donc $Q = (7, 6, 2, 9, 10)$.
- Dequeue, donc $Q = (6, 2, 9, 10)$.
- On ajoute tous les voisins non marqués de 6 à Q: aucun.
- Dequeue, donc $Q = (2, 9, 10)$.
- On ajoute tous les voisins non marqués de 2 à Q: aucun.
- Dequeue, donc $Q = (9, 10)$.
- On ajoute tous les voisins non marqués de 9 à Q: aucun.
- ...etc...

c) Nous modifions BFS comme suit (les nouvelles lignes sont les lignes 3 et 12):

Call: BFSDist(G, s)
Input: Graphe $G = (V, E)$, sommet $s \in V$.
Output: Associer à chaque sommet v une valeur $v.dist$ qui représente la distance de s à v.

```
1:  Enqueue(Q, s)
2:  Marquer s
3:  s.dist ← 0
4:  while Not QueueEmpty(Q) do
5:     v ← Head(Q)
6:     while N[v] ≠ ∅ do
7:        Choisir v' ∈ N[v]
8:        N[v] ← N[v] \ {v'}
9:        if v' n'est pas marqué then
10:          Enqueue(Q, v')
11:          Marquer v'
12:          v'.dist ← v.dist + 1
13:       end if
14:    end while
15:    Dequeue(Q)
16: end while
```

d) Nous voulons le nombre de sommets atteignables depuis s. Nous supposons que s lui-même est atteignable depuis s, nous voulons donc la taille de la composante connexe de s. Nous modifions AbstractTraversal comme suit (les nouvelles lignes sont 3,12 et 17):

Call: AbstractTraversalCount(G, s)
Input: Graphe $G = (V, E)$, sommet $s \in V$.
Output: Le nombre de sommets atteignables depuis s.

```
1:  Q ← {s}.
2:  Marquer s.
3:  count ← 1
4:  while Q ≠ ∅ do
5:     Choisir v ∈ Q.
6:     while N[v] ≠ ∅ do
```

```
7:        Choisir v' ∈ N[v]
8:        N[v] ← N[v] \ {v'}
9:        if v' n'est pas marqué then
10:         Q ← Q ∪ {v'}
11:         Marquer v'
12:         count ← count + 1
13:       end if
14:     end while
15:     Q ← Q \ {v}
16: end while
17: return count
```

En effet, il suffit de compter le nombre de sommets qui sont visités lors d'un parcours du graphe commençant à 1.

5. L'algorithme de Dijkstra

a) Appliquons l'algorithme de Dijkstra à ce graphe, pour des raisons de clarté et de comprehension de l'evolution de l'algorithme, il est indiqué le prédécesseur de chaque sommet pour une plus courte distance spécifique trouvée dans le tableau qui suit. Nous soulignons qu'il s'agit d'une indication non obligatoire.

La première colonne indique le nombre d'itérations (voir boucle while comprise entre la ligne 5 à 16 de l'algorithme dans le cours). Pour ce graphe, l'algortihme se termine au bout de 10 itérations.

La deuxième colonne indique le sommet v à chaque fois que on est au début du while (ligne 5 de l'algorithme dans le cours). En gras, nous listons les sommets, chaque ligne correspond à une itération; le contenu de chaque ligne indique les distances minimales depuis le sommet 1 vers chaque sommet de V. La dernière ligne (Fin de l'algorithme) du tableau ci-dessous indique la distance la plus courte entre 1 et chacun des sommets de V.

		Distance la plus courte (prédécesseur)											
Itér.	v	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	1	0	∞	∞	∞	6(1)	∞	∞	6(1)	∞	∞	∞	1(1)
1	12	0	∞	∞	3(12)	6(1)	∞	∞	6(1)	∞	∞	∞	1(1)
2	4	0	4(4)	∞	3(12)	6(1)	13(4)	∞	6(1)	∞	∞	∞	1(1)
3	2	0	4(4)	∞	3(12)	6(1)	6(2)	∞	6(1)	5(2)	∞	∞	1(1)
4	9	0	4(4)	∞	3(12)	6(1)	6(2)	∞	6(1)	5(2)	6(9)	∞	1(1)
5	5	0	4(4)	∞	3(12)	5(2)	6(2)	∞	6(1)	5(2)	6(9)	∞	1(1)
6	6	0	4(4)	∞	3(12)	5(2)	6(2)	∞	6(1)	5(2)	6(9)	∞	1(1)
7	8	0	4(4)	∞	3(12)	5(2)	6(2)	∞	6(1)	5(2)	6(9)	∞	1(1)
8	10	0	4(4)	∞	3(12)	5(2)	6(2)	∞	6(1)	5(2)	6(9)	9(10)	1(1)
9	11	0	4(4)	11(11)	3(12)	5(2)	6(2)	∞	6(1)	5(2)	6(9)	9(10)	1(1)

b) Nous voulons que l'algorithme nous fournisse pour chaque sommet v le plus court chemin du sommet de départ s à v et la liste des sommets qui font partie du chemin obtenu. Nous modifions DIJKSTRA(G, s) (les nouvelles lignes sont 3 et 13) comme suit:

Call: DIJKSTRA(G, s)
Input: Graphe orienté $G = (V, E)$ avec fonction de poids c: $E \to \mathbb{R}^+$, sommet $s \in V$.
Output: pour tout $v \in V$ le plus court chemin du sommet de départ s à v.

```
1:  for v ∈ V \ {s} do
2:     ℓ(v) ← ∞
3:     pred(v) = ∅
4:  end for
5:  ℓ(s) ← 0, T ← ∅, v ← s
6:  while ℓ(v) ≠ ∞ do
7:     T ← T ∪ {v}
8:     for w ∈ V \ T do
9:        if (v, w) ∈ E then
10:          d ← ℓ(v) + c(v, w)
11:          if d < ℓ(w) then
12:             ℓ(w) = d
13:             pred(w) = v
14:          end if
15:       end if
16:    end for
17:    v ← arg min_{w∈V\T} ℓ(w)
18: end while
```

Avec la version modifiée de l'algorithme de Dijkstra, les indications concernant les prédécesseurs (entre parenthèses dans le tableau) sont obligatoires pour pouvoir fournir pour chaque sommet v le plus court chemin de départ s à v.

Pour ce graphe, s vaut 1 et v vaut $2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$. En parcourant la dernière ligne de notre tableau en suivant les colonnes correspondant aux prédécesseurs, on remarque que:

(a) Le plus court chemin de 1 à 2 (qui correspond à une distance de 4) est $1 \to 12 \to 4 \to 2$.

(b) Le plus court chemin de 1 à 3 (qui correspond à une distance de 11) est $1 \to 12 \to 4 \to 2 \to 9 \to 10 \to 11 \to 3$.

(c) Le plus court chemin de 1 à 4 (qui correspond à une distance de 3) est $1 \to 12 \to 4$.

(d) Le plus court chemin de 1 à 5 (qui correspond à une distance de 6) est $1 \to 5$.

(e) Le plus court chemin de 1 à 6 (qui correspond à une distnace de 6) est $1 \to 12 \to 4 \to 2 \to 6$.

(f) Depuis 1, il n'est pas possible d'atteindre le sommet 7, d'où la valeur de ∞ dans la dernière ligne correspondant au sommet 7 du tableau précédent.

(g) Le plus court chemin de 1 à 8 (qui correspond à une distance de 6) est $1 \to 8$.

(h) Le plus court chemin de 1 à 9 (qui correspond à une distance de 5) est $1 \to 12 \to 4 \to 2 \to 9$.

(i) Le plus court chemin de 1 à 10 (qui correspond à une distance de 6) est $1 \to 12 \to 4 \to 2 \to 9 \to 10$.

(j) Le plus court chemin de 1 à 11 (qui correspond à une distance de 9) est $1 \to 12 \to 4 \to 2 \to 9 \to 10 \to 11$.

(k) Le plus court chemin de 1 à 12 (qui correspond à une distance de 1) est $1 \to 12$.