Listas de prioridades

Aula 8

Fábio Henrique Viduani Martinez Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II, Análise de Sistemas, 2010

Conteúdo da aula

Motivação

- Escalonamento de processos em um computador
- Simulação de uma lista de eventos
- Qualquer aplicação onde é associada uma prioridade a cada tarefa a ser executada

Motivação

- Escalonamento de processos em um computador
- Simulação de uma lista de eventos
- Qualquer aplicação onde é associada uma prioridade a cada tarefa a ser executada

Motivação

- Escalonamento de processos em um computador
- Simulação de uma lista de eventos
- Qualquer aplicação onde é associada uma prioridade a cada tarefa a ser executada

Heaps

Definição

Um **heap** é uma coleção de elementos identificados por suas prioridades, armazenadas em um vetor numérico S satisfazendo a seguinte propriedade:

$$S[\lfloor (i-1)/2 \rfloor] \geqslant S[i]\,, \tag{1}$$

para todo $i \geqslant 1$.

- a propriedade (1) é chamada **propriedade max-heap**
- um vetor S com a propriedade (1) é chamado um **max-heap**

Heaps

Definição

Um **heap** é uma coleção de elementos identificados por suas prioridades, armazenadas em um vetor numérico S satisfazendo a seguinte propriedade:

$$S[\lfloor (i-1)/2 \rfloor] \geqslant S[i]\,, \tag{1}$$

para todo $i \geqslant 1$.

- a propriedade (1) é chamada **propriedade max-heap**
- um vetor S com a propriedade (1) é chamado um **max-heap**

Heaps

Definição

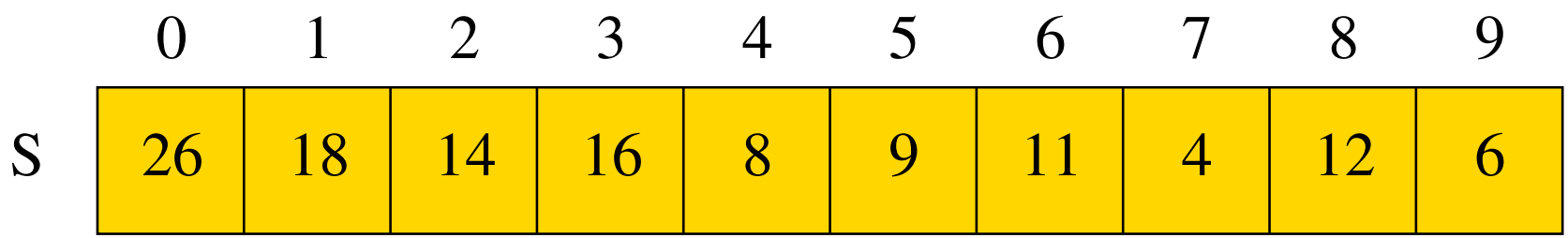
Um **heap** é uma coleção de elementos identificados por suas prioridades, armazenadas em um vetor numérico S satisfazendo a seguinte propriedade:

$$S[\lfloor (i-1)/2 \rfloor] \geqslant S[i]\,, \tag{1}$$

para todo $i \geqslant 1$.

- a propriedade (1) é chamada **propriedade max-heap**
- um vetor S com a propriedade (1) é chamado um **max-heap**

Heaps



Heaps

- é mais interessante ver um max-heap como uma árvore binária
- isso nos permite verificar a propriedade (1) mais facilmente
- o conteúdo de um nó da árvore é maior ou igual aos conteúdos dos nós que são seus filhos

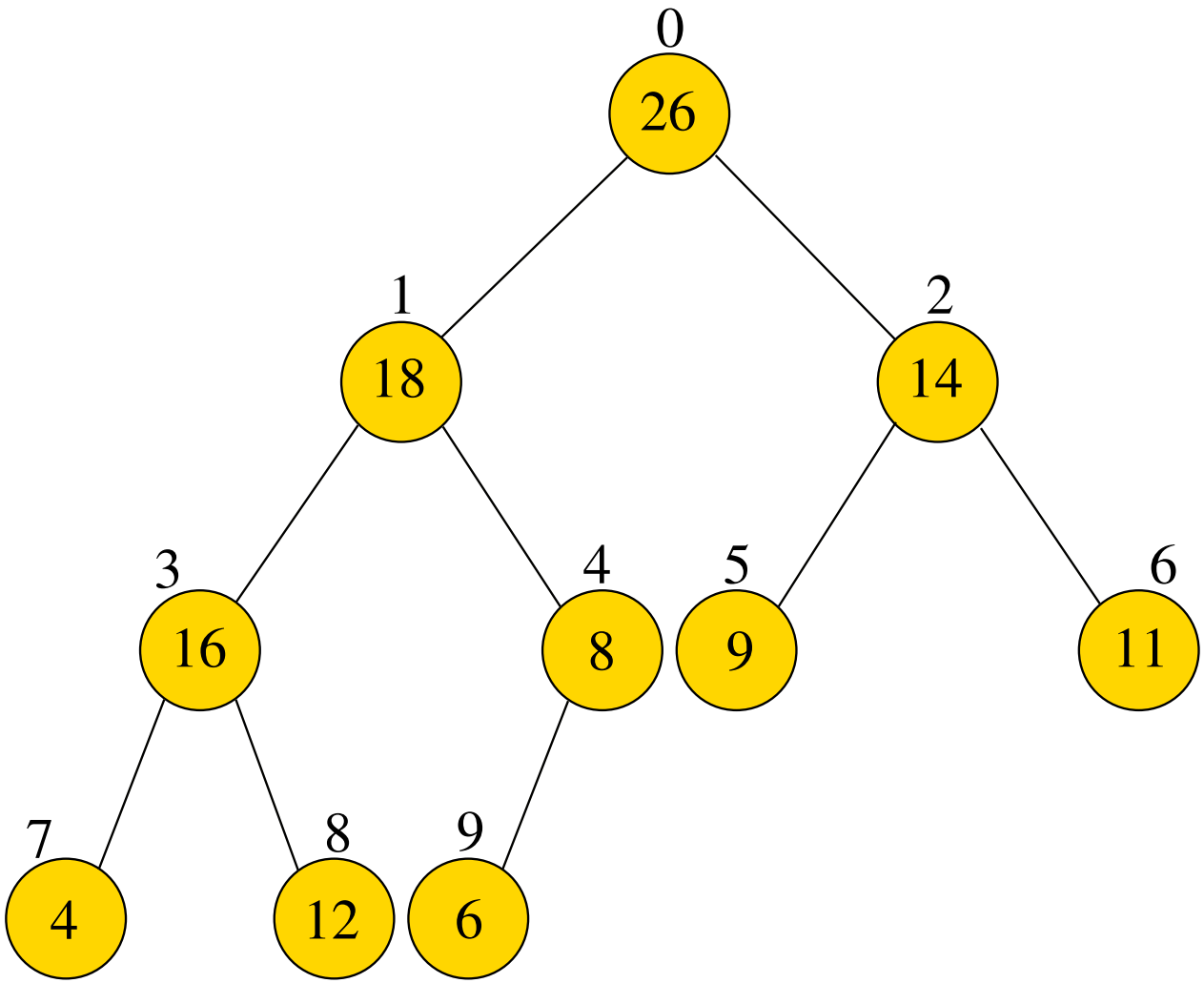
Heaps

- é mais interessante ver um max-heap como uma árvore binária
- isso nos permite verificar a propriedade (1) mais facilmente
- o conteúdo de um nó da árvore é maior ou igual aos conteúdos dos nós que são seus filhos

Heaps

- é mais interessante ver um max-heap como uma árvore binária
- isso nos permite verificar a propriedade (1) mais facilmente
- o conteúdo de um nó da árvore é maior ou igual aos conteúdos dos nós que são seus filhos

Heaps



Operações sobre max-heaps

- nó pai

```
int pai(int i)
{
   if (i == 0)
      return 0;
   else
      return (i - 1) / 2;
}
```

- filho esquerdo

```
int esquerdo(int i)
{
   return 2 * (i + 1) - 1;
}
```

- filho direito

```
int direito(int i)
{
   return 2 * (i + 1);
}
```

Operações sobre max-heaps

- nó pai

```
int pai(int i)
{
   if (i == 0)
      return 0;
   else
      return (i - 1) / 2;
}
```

- filho esquerdo

```
int esquerdo(int i)
{
   return 2 * (i + 1) - 1;
}
```

- filho direito

```
int direito(int i)
{
   return 2 * (i + 1);
}
```

Operações sobre max-heaps

- nó pai

```
int pai(int i)
{
   if (i == 0)
      return 0;
   else
      return (i - 1) / 2;
}
```

- filho esquerdo

```
int esquerdo(int i)
{
   return 2 * (i + 1) - 1;
}
```

- filho direito

```
int direito(int i)
{
   return 2 * (i + 1);
}
```

Heaps

- a propriedade max-heap (1) pode ser reescrita:

$$S[\,\texttt{pai}(i)\,] \geqslant S[i]\,, \tag{2}$$

para todo i, com $i \geqslant 0$.

Manutenção da propriedade max-heap

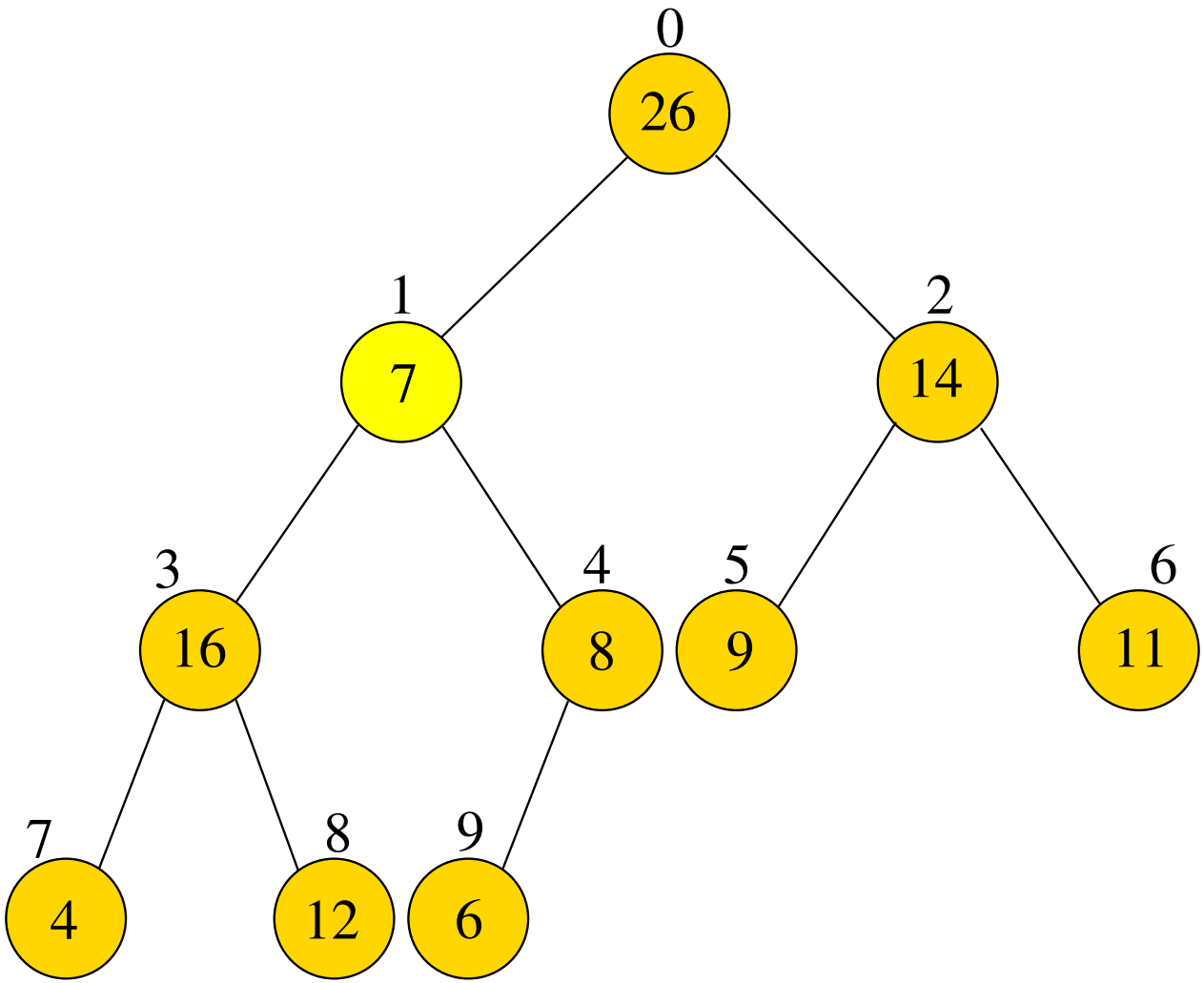
Problema

Seja um vetor de números inteiros S com $n > 0$ elementos e um índice i. Se S é visto como uma árvore binária, estabeleça a propriedade max-heap (2) para a sub-árvore de S com raiz $S[i]$, supondo que as sub-árvores esquerda e direita do nó i de S são max-heaps.

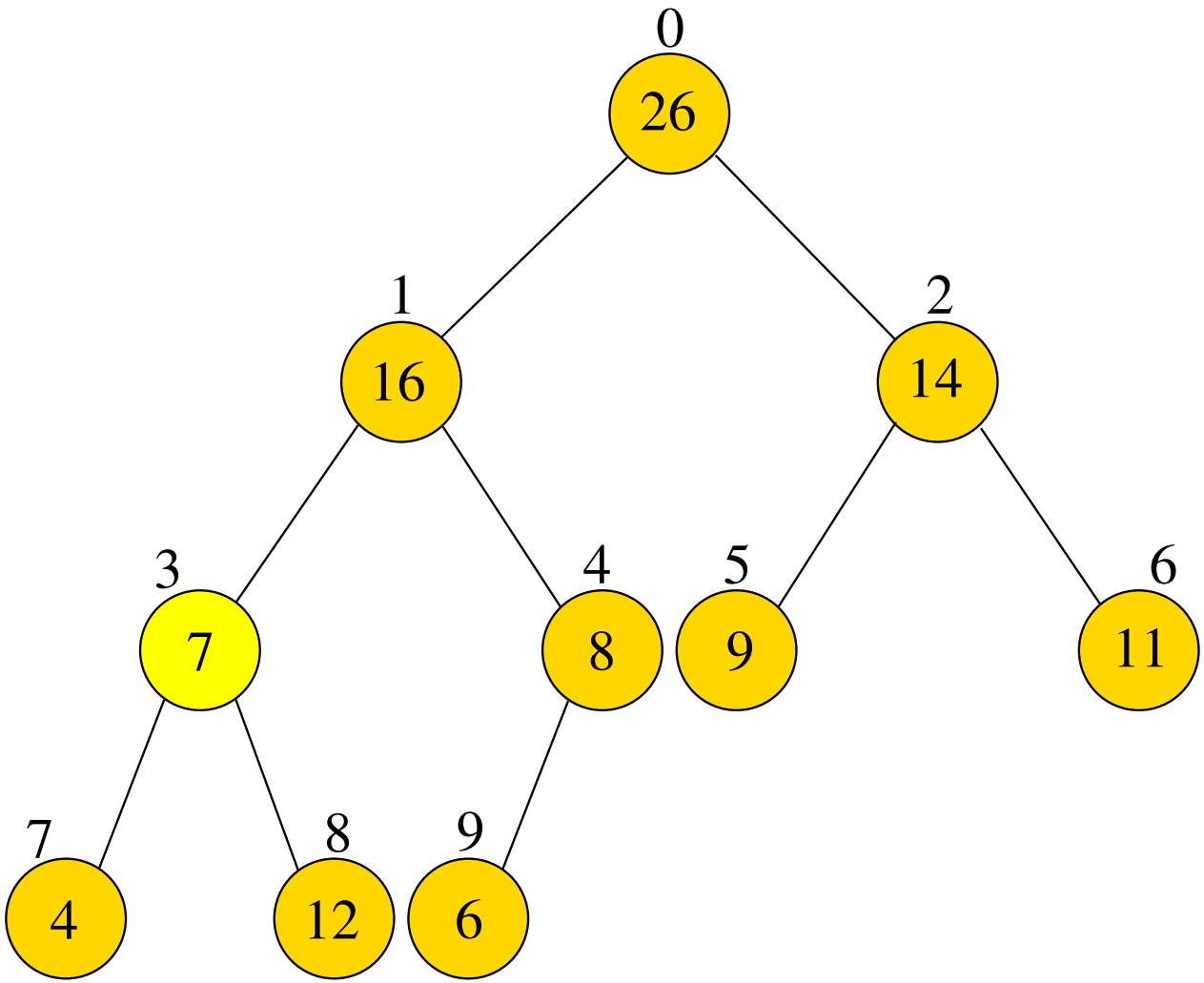
Manutenção da propriedade max-heap

```
void desce(int n, int S[MAX], int i)
{
   int e, d, maior;

   e = esquerdo(i);
   d = direito(i);
   if (e < n && S[e] > S[i])
      maior = e;
   else
      maior = i;
   if (d < n && S[d] > S[maior])
      maior = d;
   if (maior != i) {
      troca(&S[i], &S[maior]);
      desce(n, S, maior);
   }
}
```

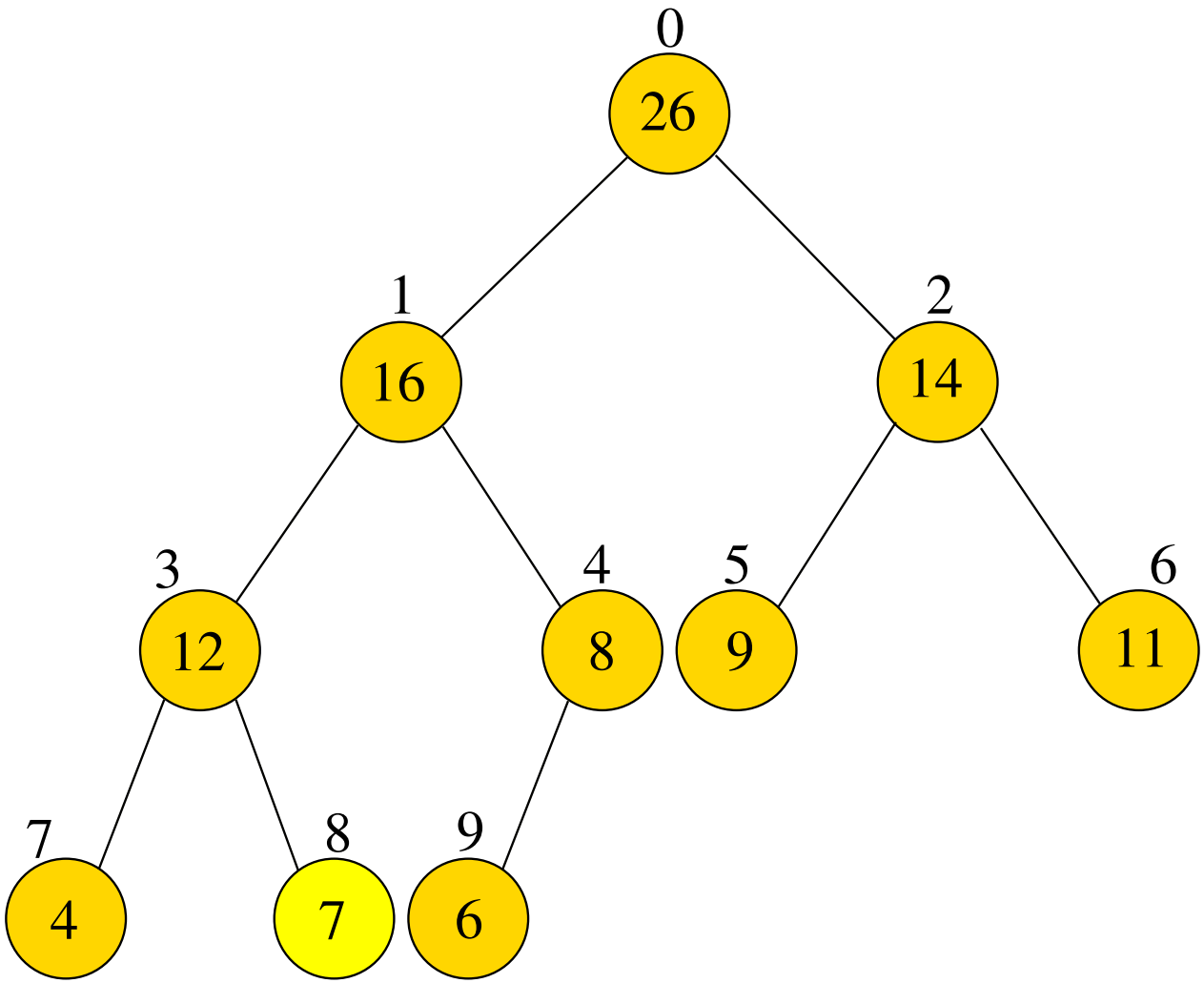
Manutenção da propriedade max-heap



Manutenção da propriedade max-heap



Manutenção da propriedade max-heap



Manutenção da propriedade max-heap

- o tempo de execução de pior caso da função `desce` é proporcional à altura da árvore binária correspondente à S, isto é, proporcional à $\log_2 n$

Construção de um max-heap

Problema

Dado um vetor S de números inteiros com $n > 0$ elementos, transformar S em um max-heap

```
void constroi_max_heap(int n, int S[MAX])
{
   int i;

   for (i = n/2 - 1; i >= 0; i--)
      desce(n, S, i);

}
```

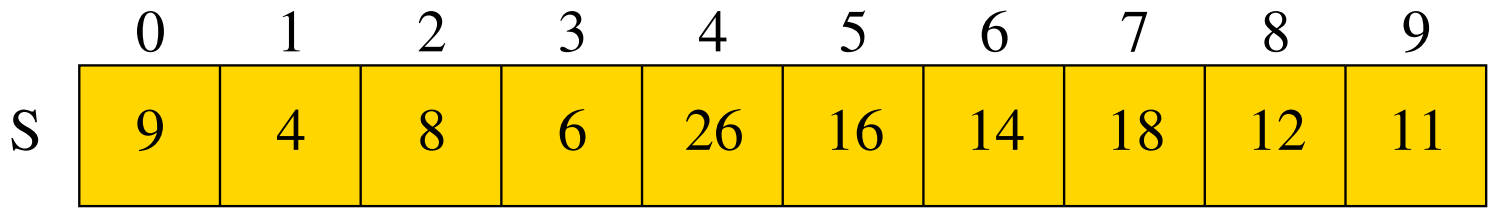
Construção de um max-heap

Problema

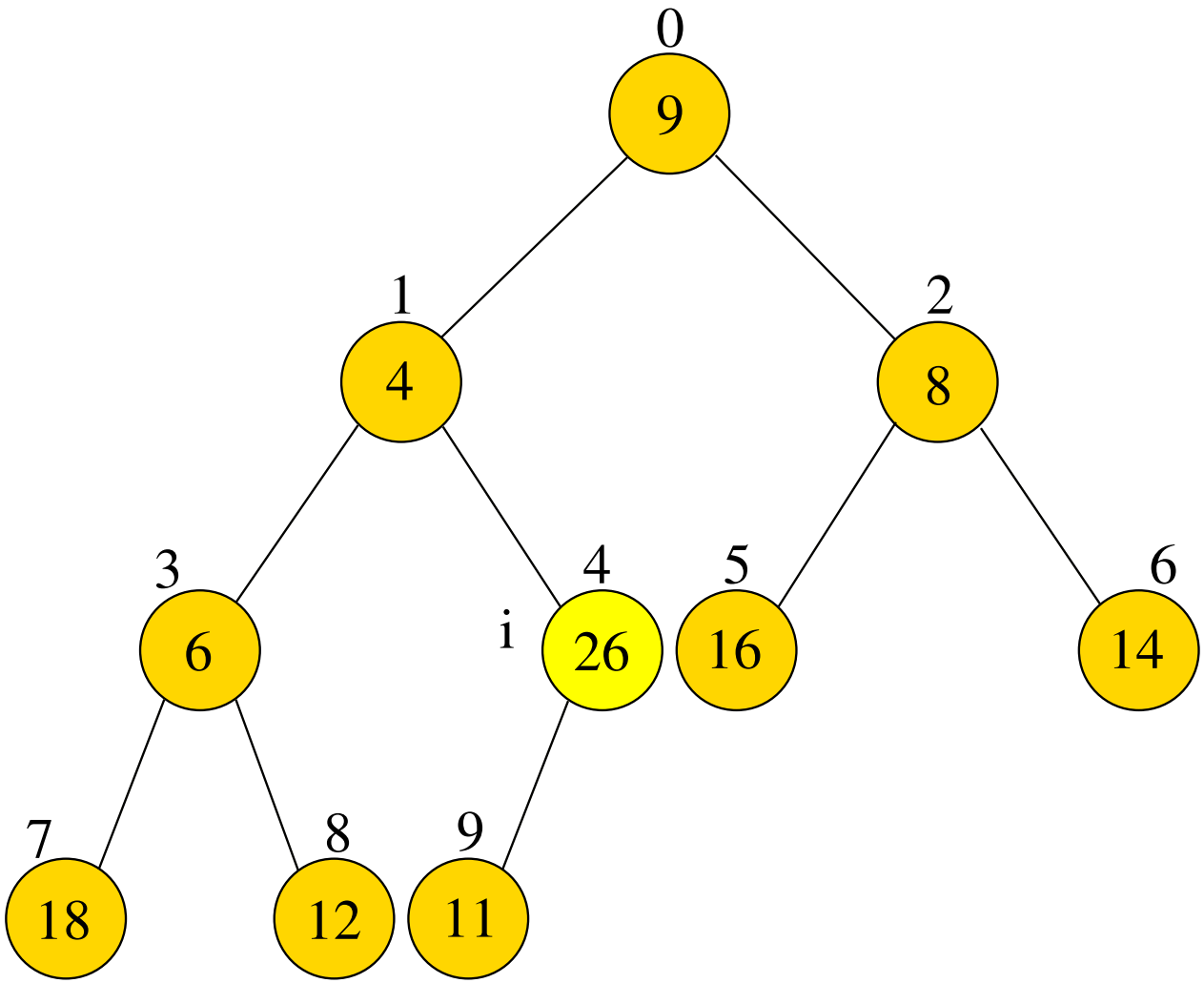
Dado um vetor S de números inteiros com $n > 0$ elementos, transformar S em um max-heap

```
void constroi_max_heap(int n, int S[MAX])
{
   int i;

   for (i = n/2 - 1; i >= 0; i--)
      desce(n, S, i);

}
```

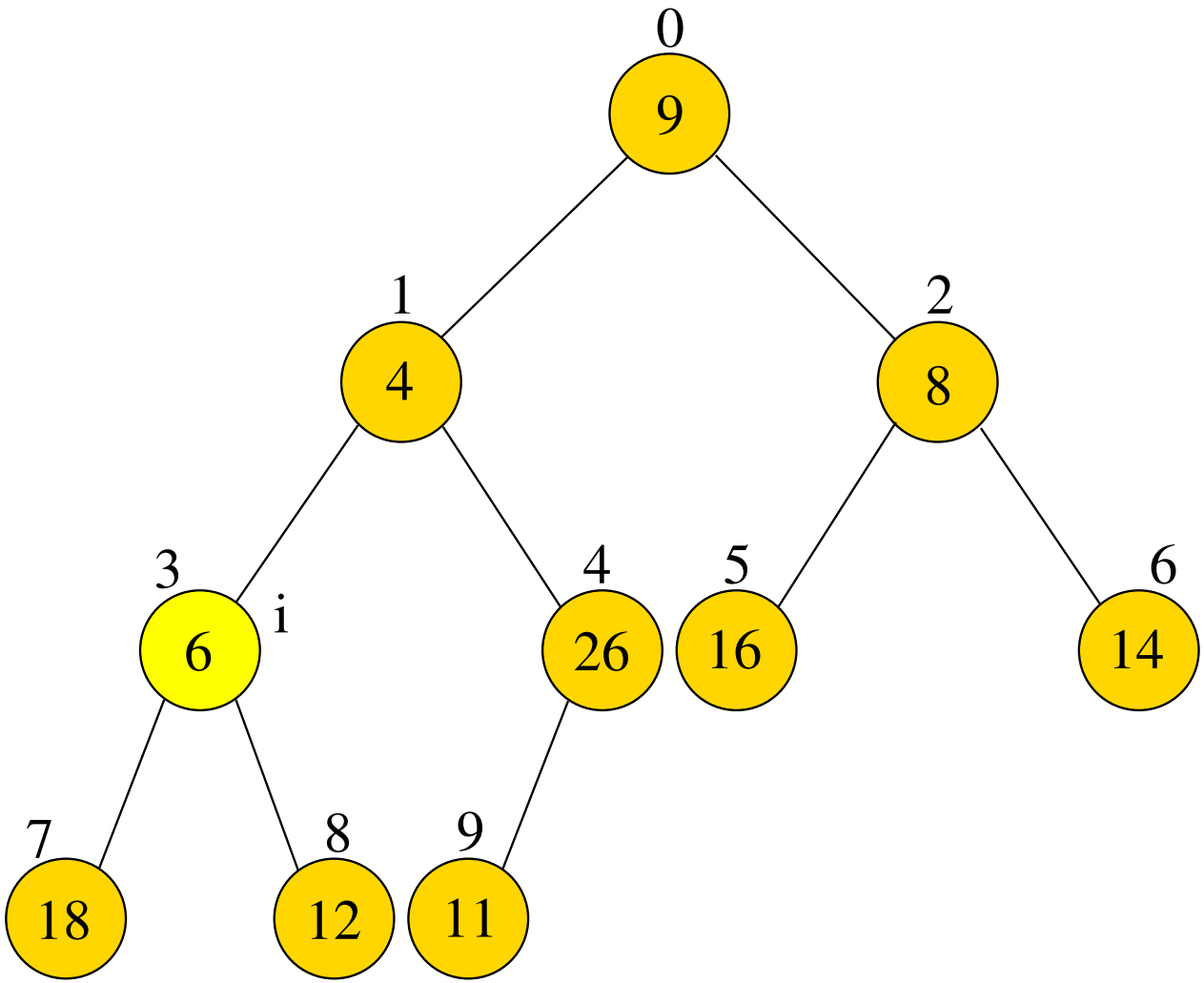
Construção de um max-heap



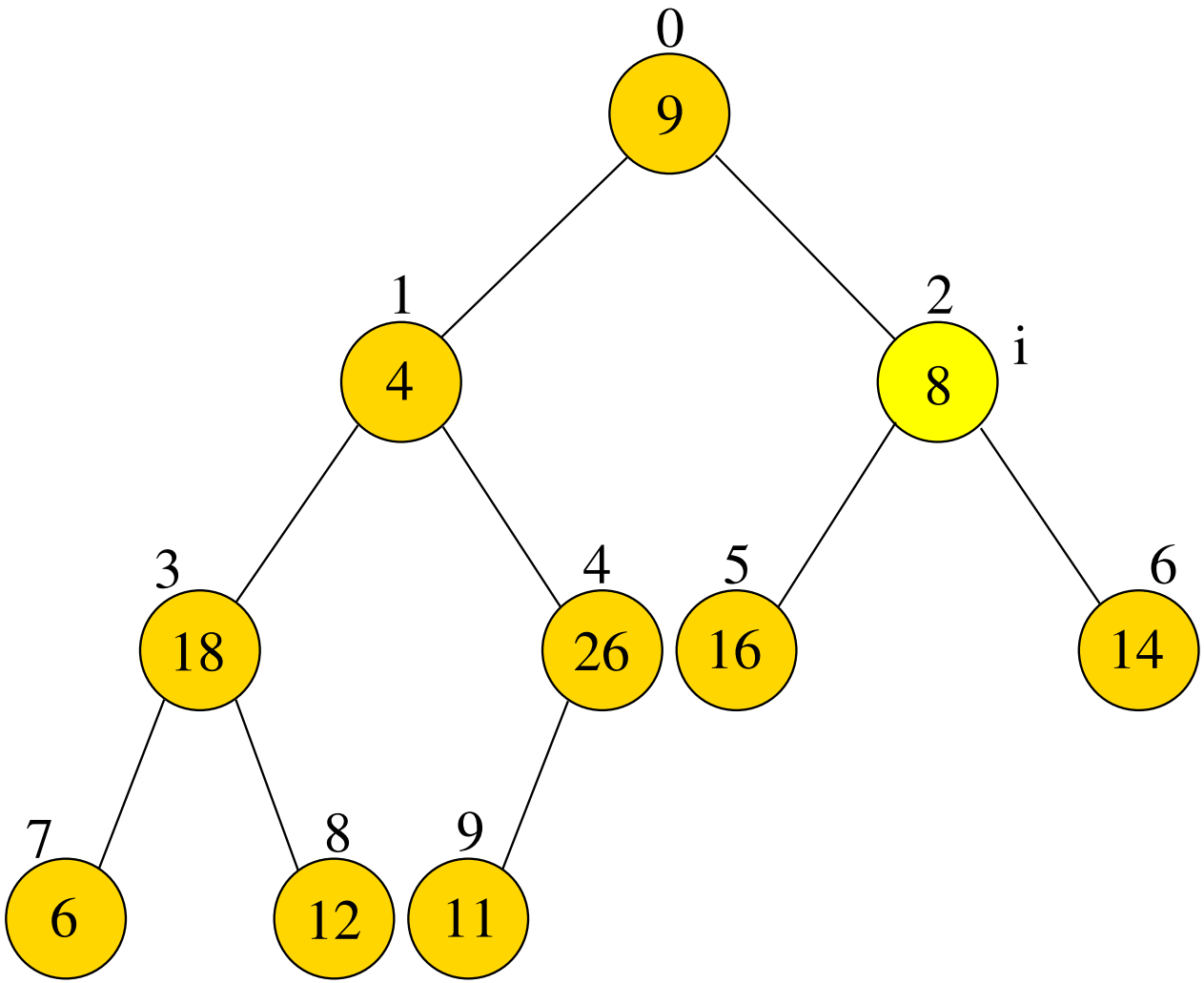
Construção de um max-heap



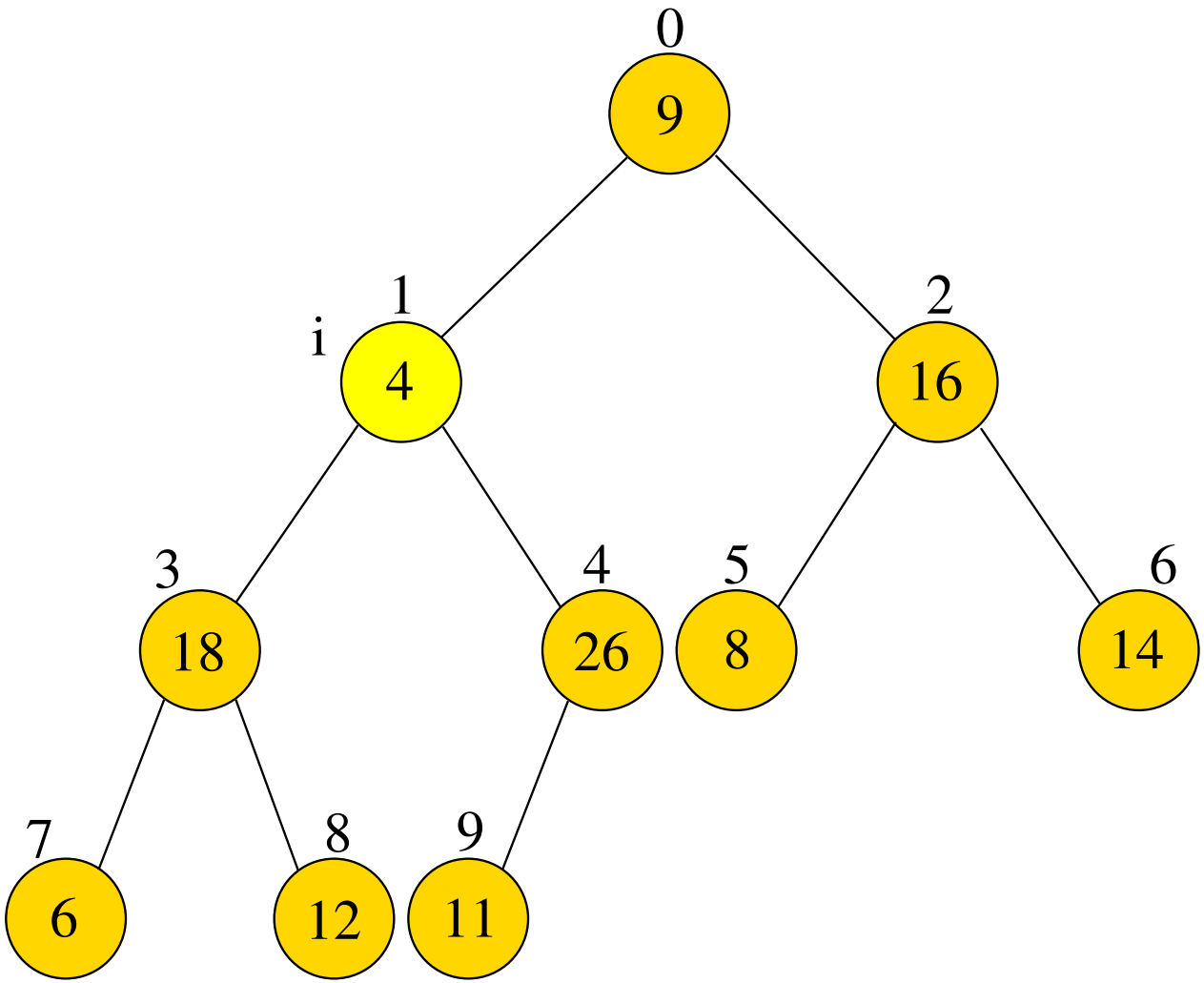
Construção de um max-heap



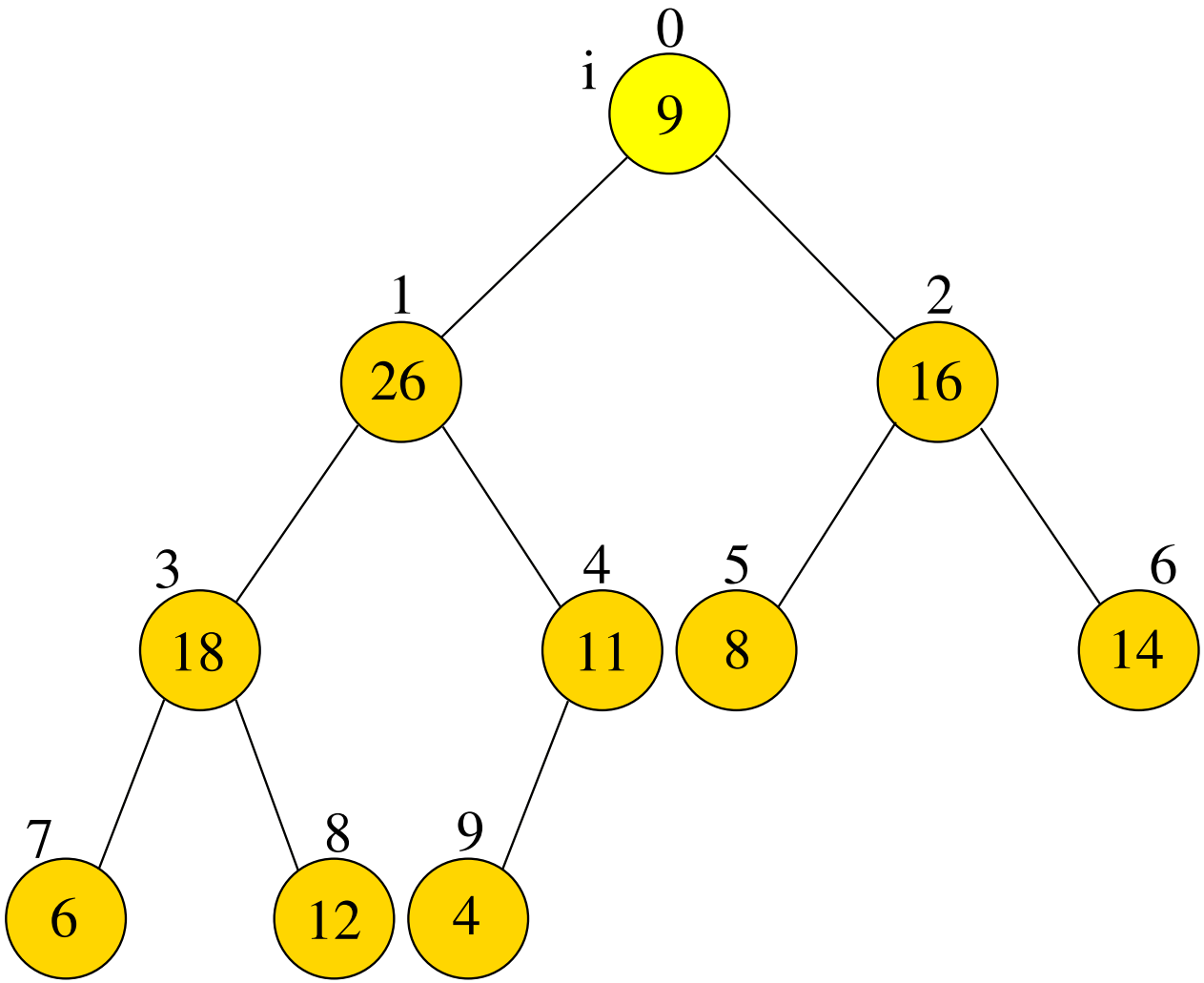
Construção de um max-heap



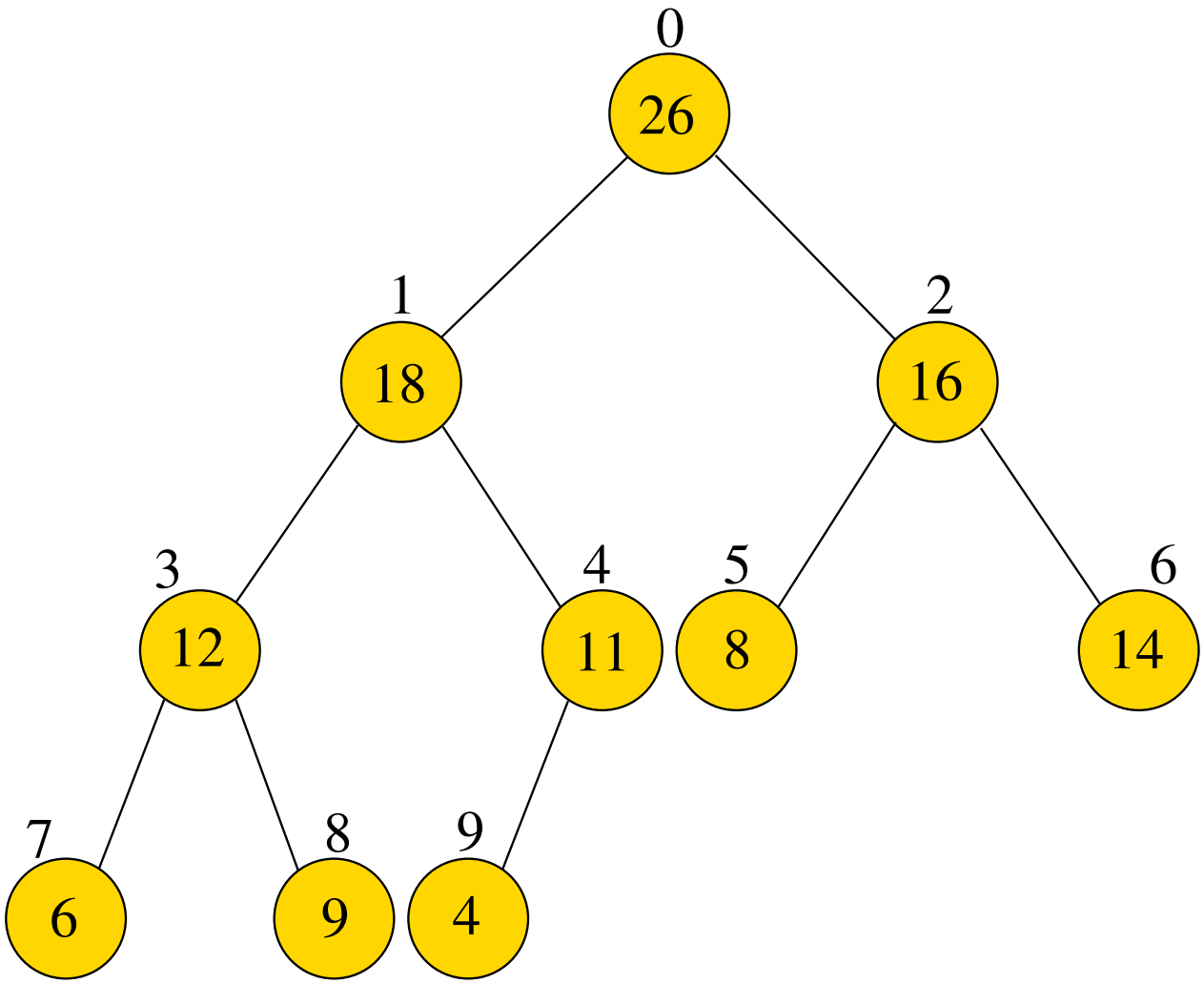
Construção de um max-heap



Construção de um max-heap



Construção de um max-heap



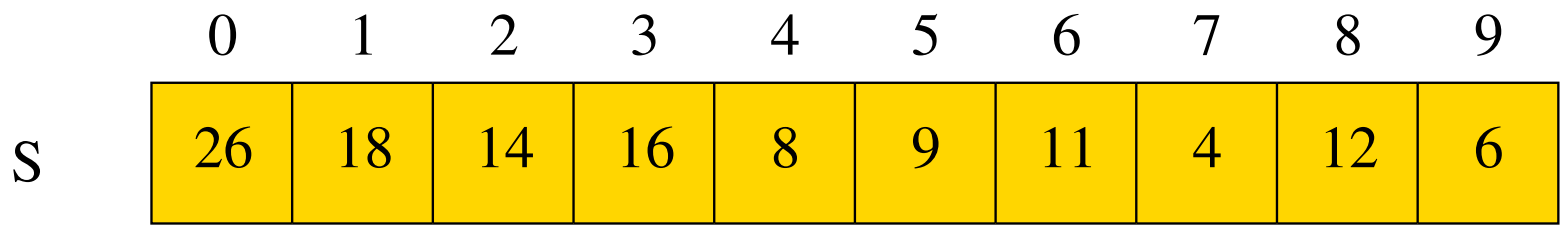
Construção de um max-heap

- Tempo de execução:
  - primeira análise: $O(n \log_2 n)$
  - análise mais apurada: $O(n)$

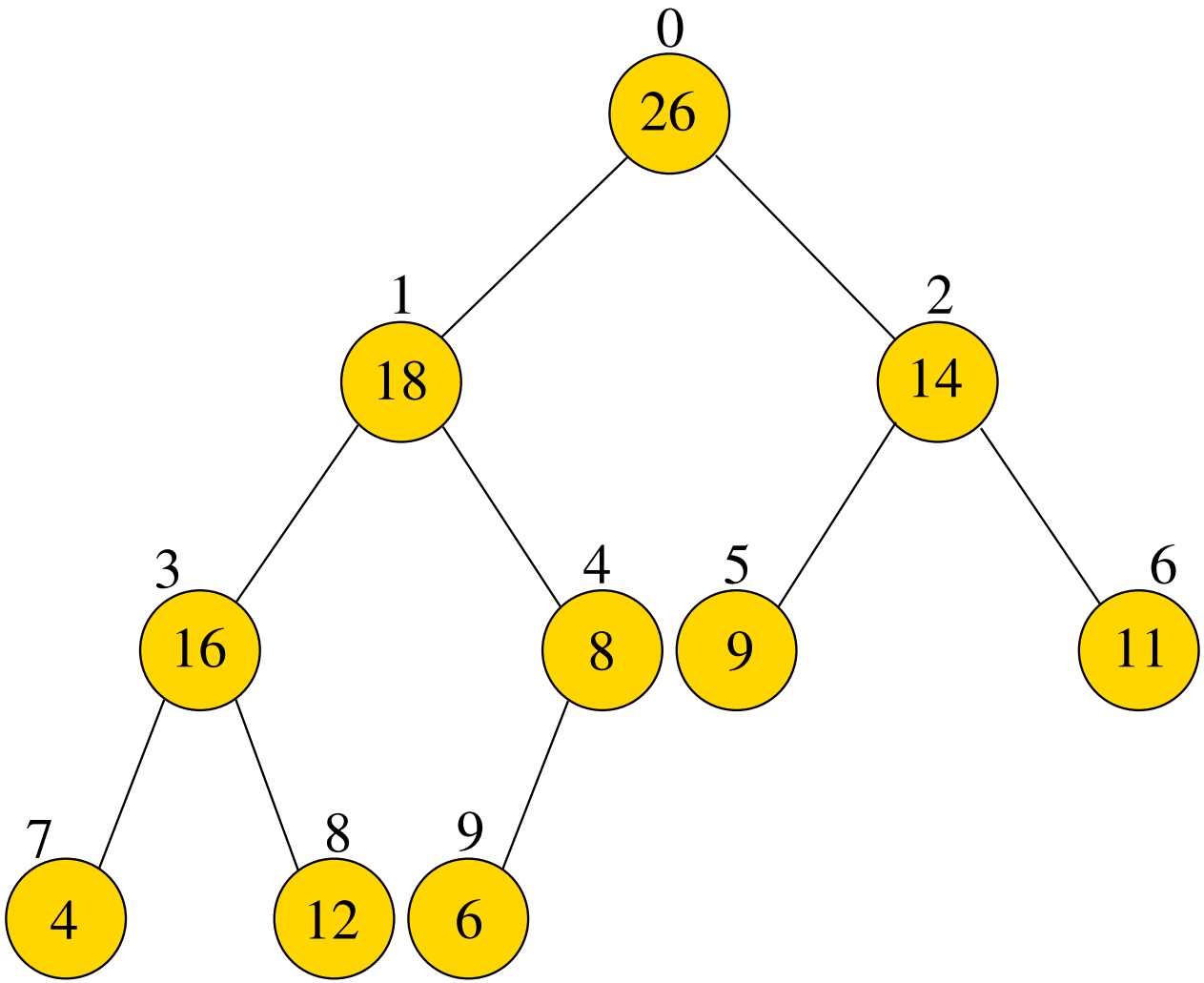
Construção de um max-heap

- Tempo de execução:
  - primeira análise: $O(n \log_2 n)$
  - análise mais apurada: $O(n)$

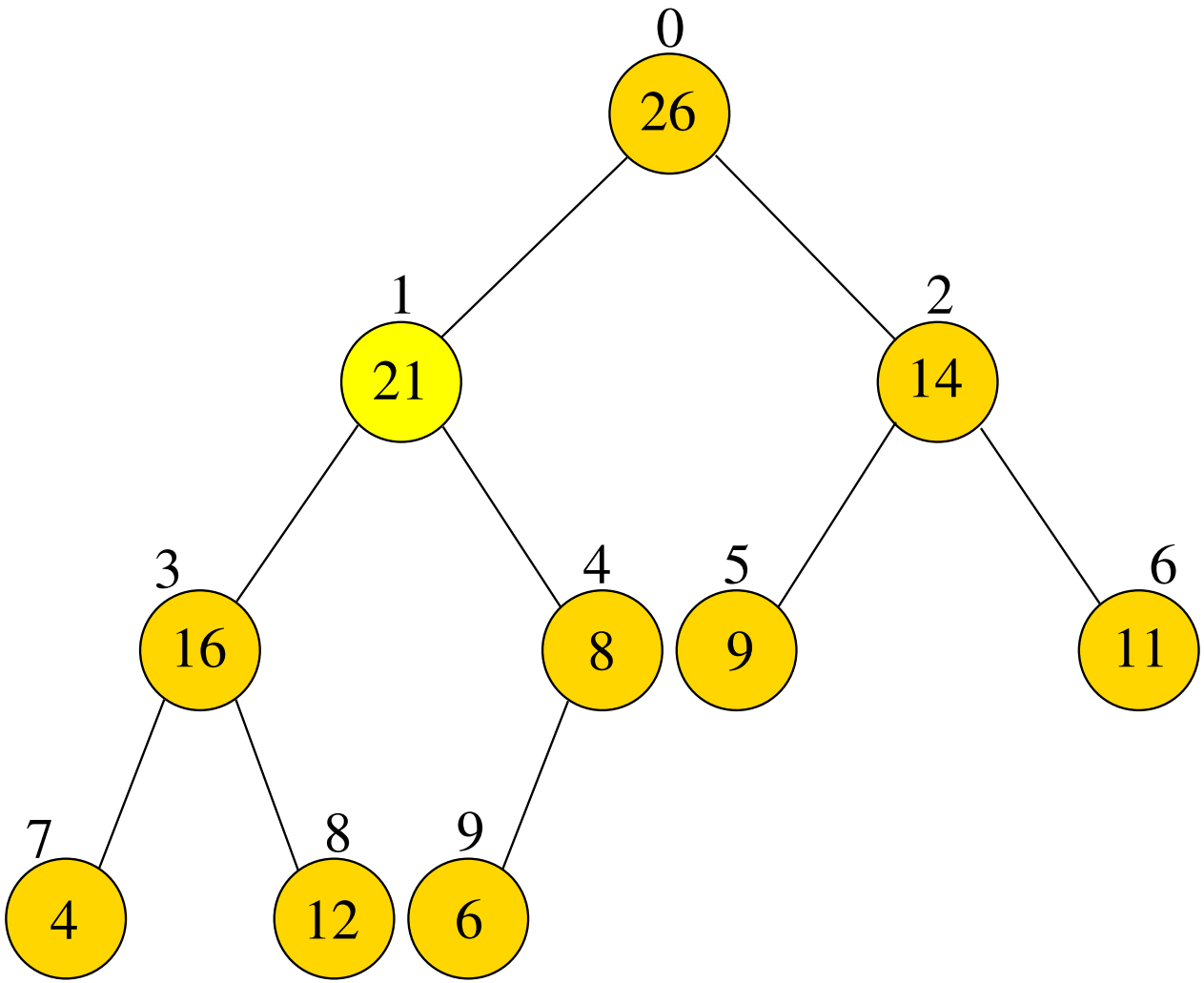
Alteração de uma prioridade em um max-heap



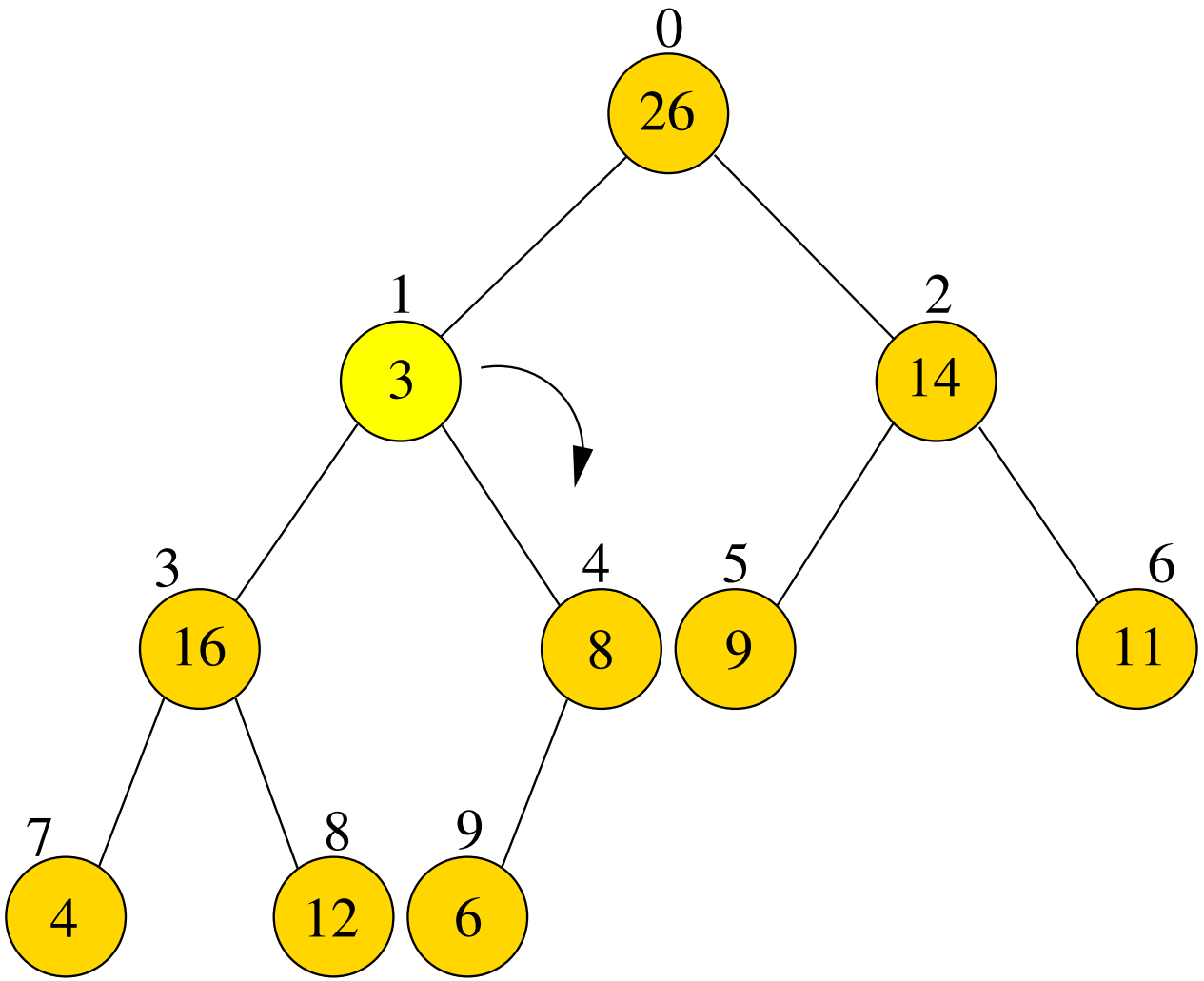
Alteração de uma prioridade em um max-heap



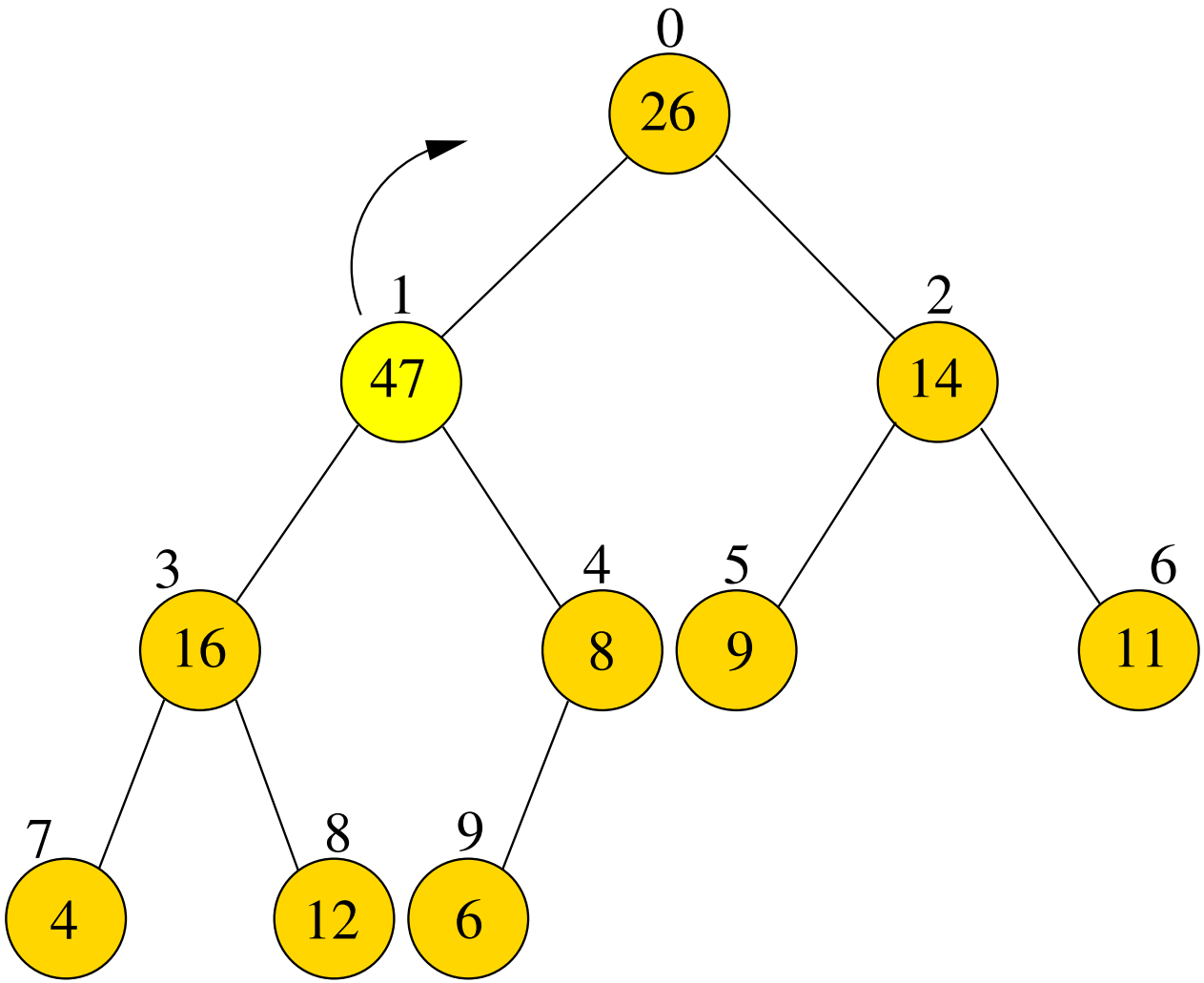
Alteração de uma prioridade em um max-heap



Alteração de uma prioridade em um max-heap



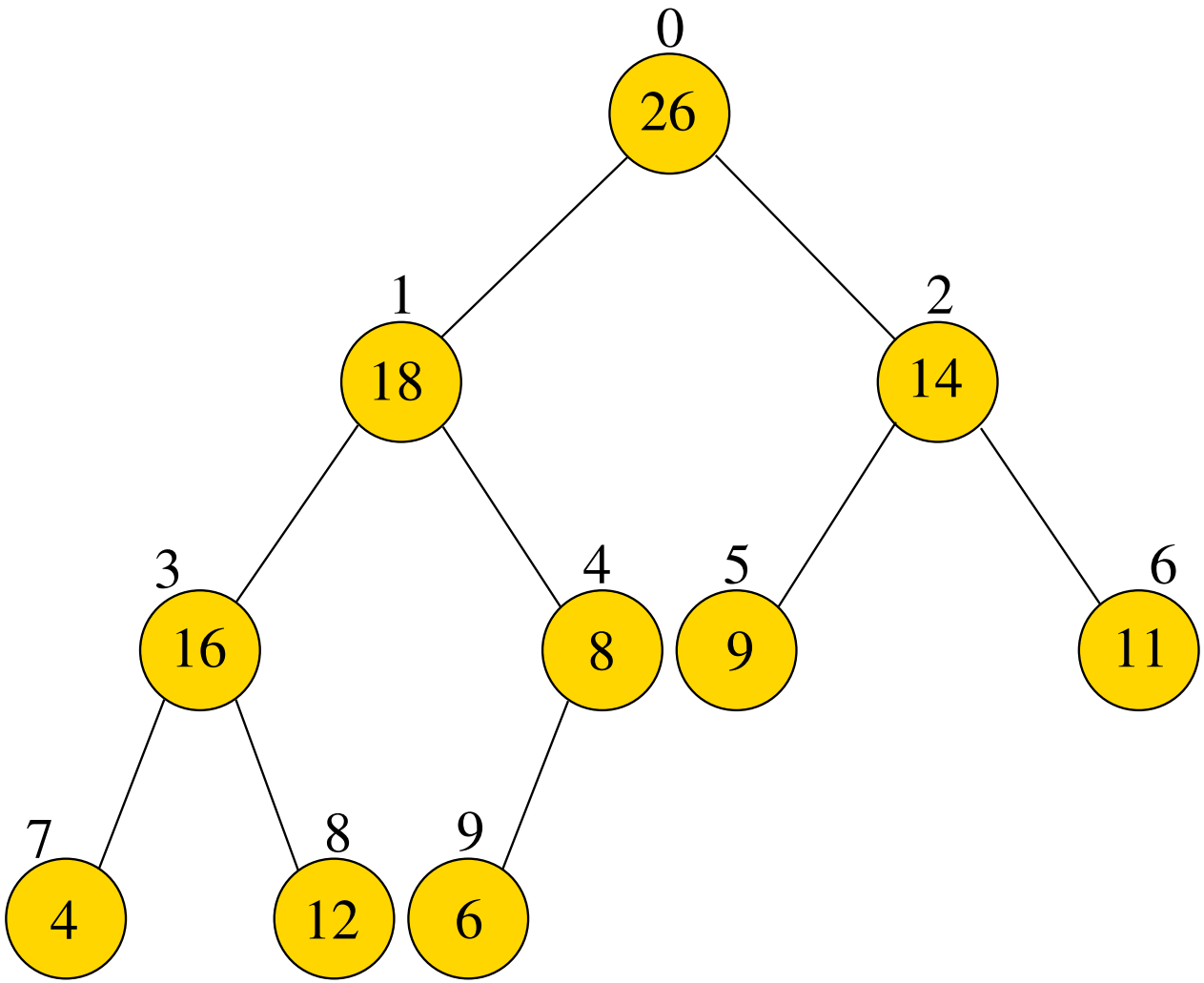
Alteração de uma prioridade em um max-heap



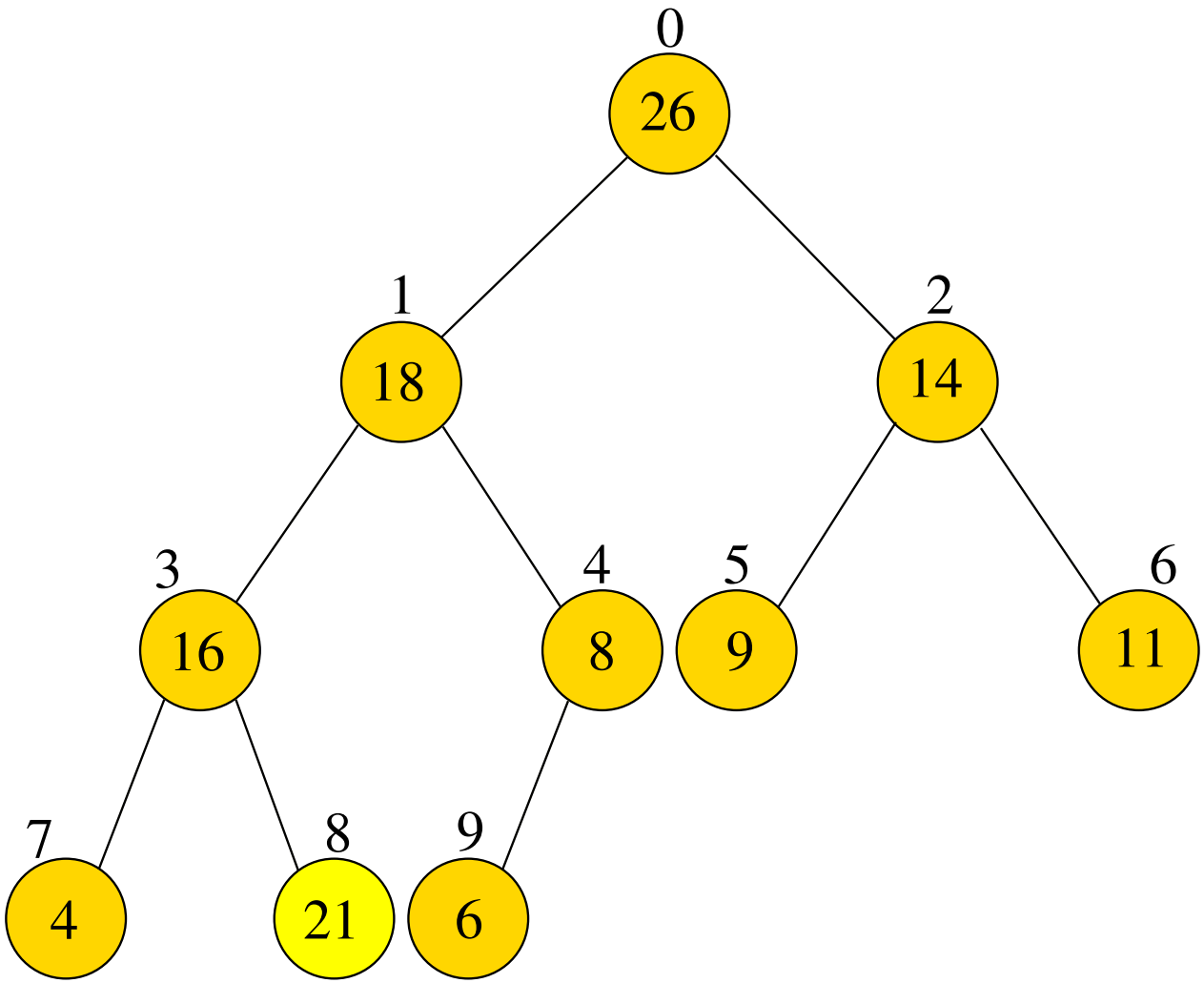
Alteração de uma prioridade em um max-heap

```
void sobe(int n, int S[MAX], int i)
{

   while (S[pai(i)] < S[i]) {
      troca(&S[i], &S[pai(i)]);
      i = pai(i);
   }

}
```

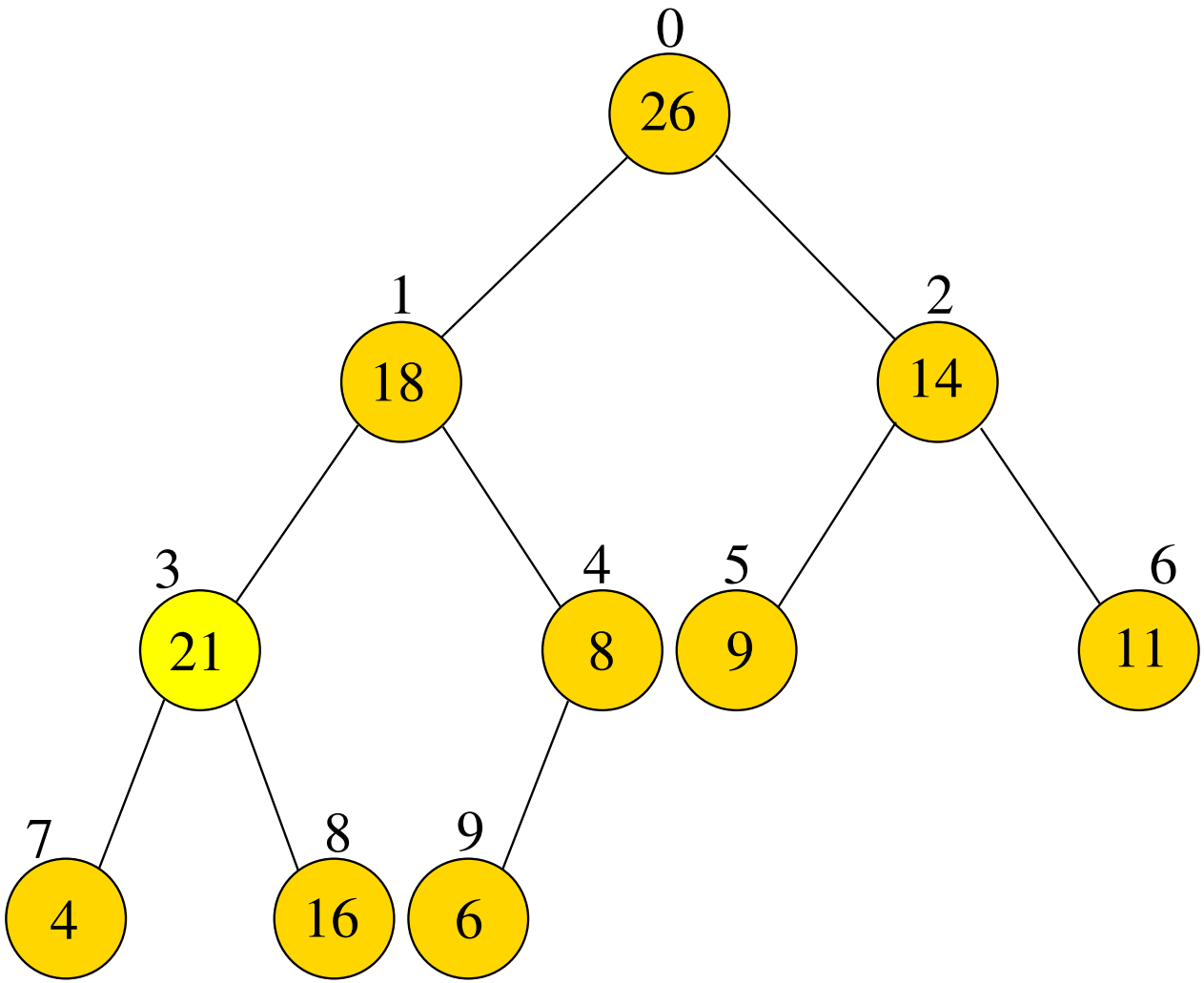
Alteração de uma prioridade em um max-heap



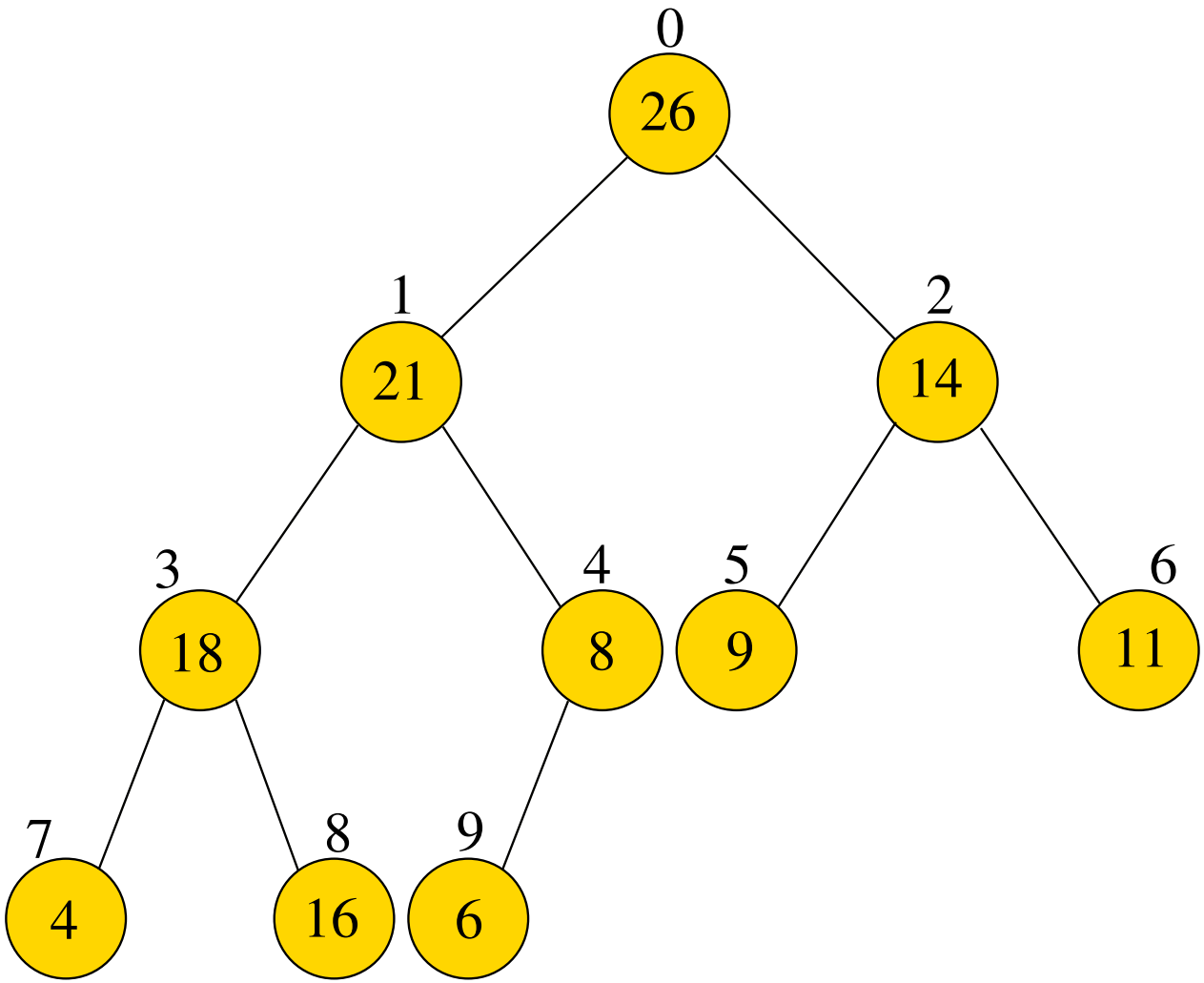
Alteração de uma prioridade em um max-heap



Alteração de uma prioridade em um max-heap



Alteração de uma prioridade em um max-heap



Alteração de uma prioridade em um max-heap

- o tempo de execução de pior caso da função `sobe` é proporcional à altura da árvore binária correspondente à S, isto é, proporcional à $\log_2 n$

Listas de prioridades

- ocorrem em muitas aplicações: escalonamento de tarefas em um computador, simulador de eventos, etc.
- listas de max-prioridades e listas de min-prioridades
- uma **lista de max-prioridades** é uma estrutura de dados para manutenção de um conjunto de elementos S, onde a cada elemento está associada uma prioridade
- As seguintes operações são associadas a uma lista de max-prioridades:
  (1) inserção de um elemento no conjunto S;
  (2) consulta da maior prioridade em S;
  (3) remoção do elemento de maior prioridade em S;
  (4) aumento da prioridade de um elemeto de S.
- um max-heap pode ser usado para implementar uma lista de max-prioridades

Listas de prioridades

- ocorrem em muitas aplicações: escalonamento de tarefas em um computador, simulador de eventos, etc.
- listas de max-prioridades e listas de min-prioridades
- uma **lista de max-prioridades** é uma estrutura de dados para manutenção de um conjunto de elementos S, onde a cada elemento está associada uma prioridade
- As seguintes operações são associadas a uma lista de max-prioridades:
  (1) inserção de um elemento no conjunto S;
  (2) consulta da maior prioridade em S;
  (3) remoção do elemento de maior prioridade em S;
  (4) aumento da prioridade de um elemeto de S.
- um max-heap pode ser usado para implementar uma lista de max-prioridades

Listas de prioridades

- ocorrem em muitas aplicações: escalonamento de tarefas em um computador, simulador de eventos, etc.
- listas de max-prioridades e listas de min-prioridades
- uma **lista de max-prioridades** é uma estrutura de dados para manutenção de um conjunto de elementos S, onde a cada elemento está associada uma prioridade
- As seguintes operações são associadas a uma lista de max-prioridades:
  (1) inserção de um elemento no conjunto S;
  (2) consulta da maior prioridade em S;
  (3) remoção do elemento de maior prioridade em S;
  (4) aumento da prioridade de um elemeto de S.
- um max-heap pode ser usado para implementar uma lista de max-prioridades

Listas de prioridades

- ocorrem em muitas aplicações: escalonamento de tarefas em um computador, simulador de eventos, etc.
- listas de max-prioridades e listas de min-prioridades
- uma **lista de max-prioridades** é uma estrutura de dados para manutenção de um conjunto de elementos S, onde a cada elemento está associada uma prioridade
- As seguintes operações são associadas a uma lista de max-prioridades:
  (1) inserção de um elemento no conjunto S;
  (2) consulta da maior prioridade em S;
  (3) remoção do elemento de maior prioridade em S;
  (4) aumento da prioridade de um elemeto de S.
- um max-heap pode ser usado para implementar uma lista de max-prioridades

Listas de prioridades

- ocorrem em muitas aplicações: escalonamento de tarefas em um computador, simulador de eventos, etc.
- listas de max-prioridades e listas de min-prioridades
- uma **lista de max-prioridades** é uma estrutura de dados para manutenção de um conjunto de elementos S, onde a cada elemento está associada uma prioridade
- As seguintes operações são associadas a uma lista de max-prioridades:
  (1) inserção de um elemento no conjunto S;
  (2) consulta da maior prioridade em S;
  (3) remoção do elemento de maior prioridade em S;
  (4) aumento da prioridade de um elemeto de S.
- um max-heap pode ser usado para implementar uma lista de max-prioridades

Listas de prioridades

- operação (2): consulta a maior prioridade em S

```
int consula_maxima(int n, int S[MAX])
{
   return S[0];
}
```

- tempo de execução constante, isto é, $O(1)$

Listas de prioridades

- operação (2): consulta a maior prioridade em S

```
int consula_maxima(int n, int S[MAX])
{
   return S[0];
}
```

- tempo de execução constante, isto é, $O(1)$

Listas de prioridades

- operação (2): consulta a maior prioridade em S

```
int consula_maxima(int n, int S[MAX])
{
   return S[0];
}
```

- tempo de execução constante, isto é, $O(1)$

Listas de prioridades

- operação (2): consulta a maior prioridade em S

```
int consula_maxima(int n, int S[MAX])
{
   return S[0];
}
```

- tempo de execução constante, isto é, $O(1)$

Listas de prioridades

- operação (3): extrai de S o elemento de prioridade máxima

```
int extrai_maxima(int *n, int S[MAX])
{
   int maior;

   if (*n > 0) {
      maior = S[0];
      S[0] = S[*n - 1];
      *n = *n - 1;
      desce(*n, S, 0);
      return maior;
   }
   else
      return -∞;
}
```

- tempo de execução da função `extrai_maxima` é na verdade o tempo gasto pela função `desce`. Portanto, seu tempo de execução é proporcional a $\log_2 n$

Listas de prioridades

- operação (3): extrai de S o elemento de prioridade máxima

```
int extrai_maxima(int *n, int S[MAX])
{
   int maior;

   if (*n > 0) {
      maior = S[0];
      S[0] = S[*n - 1];
      *n = *n - 1;
      desce(*n, S, 0);
      return maior;
   }
   else
      return -∞;
}
```

- tempo de execução da função `extrai_maxima` é na verdade o tempo gasto pela função `desce`. Portanto, seu tempo de execução é proporcional a $\log_2 n$

Listas de prioridades

- operação (3): extrai de S o elemento de prioridade máxima

```
int extrai_maxima(int *n, int S[MAX])
{
   int maior;

   if (*n > 0) {
      maior = S[0];
      S[0] = S[*n - 1];
      *n = *n - 1;
      desce(*n, S, 0);
      return maior;
   }
   else
      return -∞;
}
```

- tempo de execução da função `extrai_maxima` é na verdade o tempo gasto pela função `desce`. Portanto, seu tempo de execução é proporcional a $\log_2 n$

Listas de prioridades

- operação (3): extrai de S o elemento de prioridade máxima

```
int extrai_maxima(int *n, int S[MAX])
{
   int maior;

   if (*n > 0) {
      maior = S[0];
      S[0] = S[*n - 1];
      *n = *n - 1;
      desce(*n, S, 0);
      return maior;
   }
   else
      return -∞;
}
```

- tempo de execução da função `extrai_maxima` é na verdade o tempo gasto pela função `desce`. Portanto, seu tempo de execução é proporcional a $\log_2 n$

Listas de prioridades

- operação (4): aumento da prioridade de um elemento de S

```
void aumenta_prioridade(int n, int S[MAX], int i, int p)
{
   if (p < S[i])
      printf("ERRO: nova prioridade é menor que da célula\n");
   else {
      S[i] = p;
      sobe(n, S, i);
   }
}
```

- tempo de execução da função `aumenta_prioridade` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (4): aumento da prioridade de um elemento de S

```
void aumenta_prioridade(int n, int S[MAX], int i, int p)
{
   if (p < S[i])
      printf("ERRO: nova prioridade é menor que da célula\n");
   else {
      S[i] = p;
      sobe(n, S, i);
   }
}
```

- tempo de execução da função `aumenta_prioridade` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (4): aumento da prioridade de um elemento de S

```
void aumenta_prioridade(int n, int S[MAX], int i, int p)
{
   if (p < S[i])
      printf("ERRO: nova prioridade é menor que da célula\n");
   else {
      S[i] = p;
      sobe(n, S, i);
   }
}
```

- tempo de execução da função `aumenta_prioridade` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (4): aumento da prioridade de um elemento de S

```
void aumenta_prioridade(int n, int S[MAX], int i, int p)
{
   if (p < S[i])
      printf("ERRO: nova prioridade é menor que da célula\n");
   else {
      S[i] = p;
      sobe(n, S, i);
   }
}
```

- tempo de execução da função `aumenta_prioridade` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (1): inserção de um elemento no conjunto S

```
void insere_lista(int *n, int S[MAX], int p)
{
    *n = *n + 1;
    S[*n] = p;
    sobe(*n, S, *n - 1);
}
```

- tempo de execução da função `insere_lista` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (1): inserção de um elemento no conjunto S

```
void insere_lista(int *n, int S[MAX], int p)
{
    *n = *n + 1;
    S[*n] = p;
    sobe(*n, S, *n - 1);
}
```

- tempo de execução da função `insere_lista` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (1): inserção de um elemento no conjunto S

```
void insere_lista(int *n, int S[MAX], int p)
{
    *n = *n + 1;
    S[*n] = p;
    sobe(*n, S, *n - 1);
}
```

- tempo de execução da função `insere_lista` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Listas de prioridades

- operação (1): inserção de um elemento no conjunto S

```
void insere_lista(int *n, int S[MAX], int p)
{
    *n = *n + 1;
    S[*n] = p;
    sobe(*n, S, *n - 1);
}
```

- tempo de execução da função `insere_lista` é o tempo gasto pela chamada à função `sobe` e, portanto, é proporcional a $\log_2 n$

Ordenação usando um max-heap

- um max-heap pode ser naturalmente usado para descrever um algoritmo de ordenação eficiente
- algoritmo conhecido como *heapsort*
- mesmo tempo de execução de pior caso da ordenação por intercalação e do caso médio da ordenação por separação

Ordenação usando um max-heap

- um max-heap pode ser naturalmente usado para descrever um algoritmo de ordenação eficiente
- algoritmo conhecido como *heapsort*
- mesmo tempo de execução de pior caso da ordenação por intercalação e do caso médio da ordenação por separação

Ordenação usando um max-heap

- um max-heap pode ser naturalmente usado para descrever um algoritmo de ordenação eficiente
- algoritmo conhecido como *heapsort*
- mesmo tempo de execução de pior caso da ordenação por intercalação e do caso médio da ordenação por separação

Ordenação usando um max-heap

```
void heapsort(int n, int S[MAX])
{
   int i;

   constroi_max_heap(n, S);
   for (i = n - 1; i > 0; i--) {
      troca(&S[0], &S[i]);
      n--;
      desce(n, S, 0);
   }
}
```

Ordenação usando um max-heap

- a função `heapsort` está correta devido ao seguinte invariante do processo iterativo:

  No início de cada iteração da estrutura de repetição da função `heapsort`*, o vetor* $S[0..i]$ *é um max-heap contendo os* i *menores elementos de* $S[0..n-1]$ *e o vetor* $S[i+1..n-1]$ *contém os* $n-i$ *maiores elementos de* $S[0..n-1]$ *em ordem crescente*

- tempo de execução da função `heapsort` é proporcional a $n \log_2 n$

Ordenação usando um max-heap

- a função `heapsort` está correta devido ao seguinte invariante do processo iterativo:

  No início de cada iteração da estrutura de repetição da função `heapsort`*, o vetor* $S[0..i]$ *é um max-heap contendo os* i *menores elementos de* $S[0..n-1]$ *e o vetor* $S[i+1..n-1]$ *contém os* $n-i$ *maiores elementos de* $S[0..n-1]$ *em ordem crescente*

- tempo de execução da função `heapsort` é proporcional a $n \log_2 n$

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.1 A seqüência $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ é um max-heap?

8.2 Qual são os números mínimo e máximo de elementos em um max-heap de altura h?

8.3 Mostre que em qualquer sub-árvore de um max-heap, a raiz da sub-árvore contém a maior prioridade de todas as que ocorrem naquela sub-árvore.

8.4 Em um max-heap, onde pode estar armazenado o elemento de menor prioridade, considerando que todos os elementos são distintos?

8.5 Um vetor em ordem crescente é um min-heap?

8.6 Ilustre a execução da função `desce(14, S, 2)` sobre o vetor $S = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$.

8.7 Suponha que você deseja manter um min-heap. Escreva uma função equivalente à função `desce` para um max-heap, que mantém a propriedade min-heap.

Exercícios

8.8 Qual o efeito de chamar `desce(n, S, i)` quando a prioridade $S[i]$ é maior que as prioridades de seus filhos?

8.9 Qual o efeito de chamar `desce(n, S, i)` para $i \geqslant n/2$?

8.10 O código da função `desce` é muito eficiente em termos de fatores constantes, exceto possivelmente pela chamada recursiva que pode fazer com que alguns compiladores produzam um código ineficiente. Escreva uma função não-recursiva eficiente equivalente à função `desce`.

8.11 Ilustre a operação da função `constroi_max_heap` sobre o vetor $S = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

8.12 Por que fazemos com que a estrutura de repetição da função `constroi_max_heap` controlada por i seja decrescente de $n/2 - 1$ até 0 ao invés de crescente de 0 até $n/2 - 1$?

8.13 Ilustre a operação da função `extrai_maximo` sobre o vetor $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

Exercícios

8.8 Qual o efeito de chamar `desce(n, S, i)` quando a prioridade $S[i]$ é maior que as prioridades de seus filhos?

8.9 Qual o efeito de chamar `desce(n, S, i)` para $i \geqslant n/2$?

8.10 O código da função `desce` é muito eficiente em termos de fatores constantes, exceto possivelmente pela chamada recursiva que pode fazer com que alguns compiladores produzam um código ineficiente. Escreva uma função não-recursiva eficiente equivalente à função `desce`.

8.11 Ilustre a operação da função `constroi_max_heap` sobre o vetor $S = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

8.12 Por que fazemos com que a estrutura de repetição da função `constroi_max_heap` controlada por i seja decrescente de $n/2 - 1$ até 0 ao invés de crescente de 0 até $n/2 - 1$?

8.13 Ilustre a operação da função `extrai_maximo` sobre o vetor $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

Exercícios

8.8 Qual o efeito de chamar `desce(n, S, i)` quando a prioridade $S[i]$ é maior que as prioridades de seus filhos?

8.9 Qual o efeito de chamar `desce(n, S, i)` para $i \geqslant n/2$?

8.10 O código da função `desce` é muito eficiente em termos de fatores constantes, exceto possivelmente pela chamada recursiva que pode fazer com que alguns compiladores produzam um código ineficiente. Escreva uma função não-recursiva eficiente equivalente à função `desce`.

8.11 Ilustre a operação da função `constroi_max_heap` sobre o vetor $S = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

8.12 Por que fazemos com que a estrutura de repetição da função `constroi_max_heap` controlada por i seja decrescente de $n/2 - 1$ até 0 ao invés de crescente de 0 até $n/2 - 1$?

8.13 Ilustre a operação da função `extrai_maximo` sobre o vetor $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

Exercícios

8.8 Qual o efeito de chamar `desce(n, S, i)` quando a prioridade $S[i]$ é maior que as prioridades de seus filhos?

8.9 Qual o efeito de chamar `desce(n, S, i)` para $i \geqslant n/2$?

8.10 O código da função `desce` é muito eficiente em termos de fatores constantes, exceto possivelmente pela chamada recursiva que pode fazer com que alguns compiladores produzam um código ineficiente. Escreva uma função não-recursiva eficiente equivalente à função `desce`.

8.11 Ilustre a operação da função `constroi_max_heap` sobre o vetor $S = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

8.12 Por que fazemos com que a estrutura de repetição da função `constroi_max_heap` controlada por i seja decrescente de $n/2 - 1$ até 0 ao invés de crescente de 0 até $n/2 - 1$?

8.13 Ilustre a operação da função `extrai_maximo` sobre o vetor $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

Exercícios

8.8 Qual o efeito de chamar `desce(n, S, i)` quando a prioridade $S[i]$ é maior que as prioridades de seus filhos?

8.9 Qual o efeito de chamar `desce(n, S, i)` para $i \geqslant n/2$?

8.10 O código da função `desce` é muito eficiente em termos de fatores constantes, exceto possivelmente pela chamada recursiva que pode fazer com que alguns compiladores produzam um código ineficiente. Escreva uma função não-recursiva eficiente equivalente à função `desce`.

8.11 Ilustre a operação da função `constroi_max_heap` sobre o vetor $S = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

8.12 Por que fazemos com que a estrutura de repetição da função `constroi_max_heap` controlada por i seja decrescente de $n/2 - 1$ até 0 ao invés de crescente de 0 até $n/2 - 1$?

8.13 Ilustre a operação da função `extrai_maximo` sobre o vetor $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

Exercícios

8.8 Qual o efeito de chamar `desce(n, S, i)` quando a prioridade $S[i]$ é maior que as prioridades de seus filhos?

8.9 Qual o efeito de chamar `desce(n, S, i)` para $i \geqslant n/2$?

8.10 O código da função `desce` é muito eficiente em termos de fatores constantes, exceto possivelmente pela chamada recursiva que pode fazer com que alguns compiladores produzam um código ineficiente. Escreva uma função não-recursiva eficiente equivalente à função `desce`.

8.11 Ilustre a operação da função `constroi_max_heap` sobre o vetor $S = \langle 5, 3, 17, 10, 84, 19, 6, 22, 9 \rangle$.

8.12 Por que fazemos com que a estrutura de repetição da função `constroi_max_heap` controlada por i seja decrescente de $n/2 - 1$ até 0 ao invés de crescente de 0 até $n/2 - 1$?

8.13 Ilustre a operação da função `extrai_maximo` sobre o vetor $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

Exercícios

8.14 Ilustre a operação da função `insere_lista(12, S, 9)` sobre a lista de prioridades $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

8.15 Escreva códigos eficientes e corretos para as funções que implementam as operações `consulta_minimo`, `extrai_minimo`, `diminui_prioridade` e `insere_lista_min`. Essas funções devem implementar uma lista de min-prioridades com um min-heap.

8.16 A operação `remove_lista(&n, S, i)` remove a prioridade do nó i de uma lista de max-prioridades. Escreva uma função eficiente para `remove_lista`.

8.17 Ilustre a execução da função `heapsort` sobre o vetor $S = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.

Exercícios

8.14 Ilustre a operação da função `insere_lista(12, S, 9)` sobre a lista de prioridades $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

8.15 Escreva códigos eficientes e corretos para as funções que implementam as operações `consulta_minimo`, `extrai_minimo`, `diminui_prioridade` e `insere_lista_min`. Essas funções devem implementar uma lista de min-prioridades com um min-heap.

8.16 A operação `remove_lista(&n, S, i)` remove a prioridade do nó i de uma lista de max-prioridades. Escreva uma função eficiente para `remove_lista`.

8.17 Ilustre a execução da função `heapsort` sobre o vetor $S = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.

Exercícios

8.14 Ilustre a operação da função `insere_lista(12, S, 9)` sobre a lista de prioridades $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

8.15 Escreva códigos eficientes e corretos para as funções que implementam as operações `consulta_minimo`, `extrai_minimo`, `diminui_prioridade` e `insere_lista_min`. Essas funções devem implementar uma lista de min-prioridades com um min-heap.

8.16 A operação `remove_lista(&n, S, i)` remove a prioridade do nó i de uma lista de max-prioridades. Escreva uma função eficiente para `remove_lista`.

8.17 Ilustre a execução da função `heapsort` sobre o vetor $S = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.

Exercícios

8.14 Ilustre a operação da função `insere_lista(12, S, 9)` sobre a lista de prioridades $S = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$.

8.15 Escreva códigos eficientes e corretos para as funções que implementam as operações `consulta_minimo`, `extrai_minimo`, `diminui_prioridade` e `insere_lista_min`. Essas funções devem implementar uma lista de min-prioridades com um min-heap.

8.16 A operação `remove_lista(&n, S, i)` remove a prioridade do nó i de uma lista de max-prioridades. Escreva uma função eficiente para `remove_lista`.

8.17 Ilustre a execução da função `heapsort` sobre o vetor $S = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$.