Algorytmy i struktury danych - Algorytmy z powrotami

Marcin Żurowski

12 czerwca 2025

Algorytmy z powrotami

Algorytmy z powrotami wykorzystujemy do rozwiązywania problemów, w których z określonego zbioru obiektów jest wybierana ich sekwencja spełniająca określone kryteria.

Algorytm z powrotami można interpretować jako przeszukiwanie w głąb drzewa reprezentującego tworzenie możliwych sekwencji obiektów (drzewa przestrzeni stanów):

- Zaczynamy od korzenia drzewa
- Dla każdego poddrzewa, które rozważamy, jego korzeń jest następnie rekurencyjnie przeszukiwany w celu znalezienia
- Odwiedzenie liścia drzewa oznacza wygenerowanie całej sekwencji obiektów (potencjalnego rozwiązania problemu)

Algorytmy z powrotami

Algorytmy z powrotami wykorzystujemy do rozwiązywania problemów, w których z określonego zbioru obiektów jest wybierana ich sekwencja spełniająca określone kryteria.
Algorytm z powrotami można interpretować jako przeszukiwanie w głąb drzewa reprezentującego tworzenie możliwych sekwencji obiektów (drzewa przestrzeni stanów):

- Zaczynamy od korzenia drzewa
- Dla każdego poddrzewa najpierw odwiedzamy jego korzeń, a następnie rekurencyjnie przeszukujemy wszystkich jego synów
- Odwiedzenie liścia drzewa oznacza wygenerowanie pełnej sekwencji obiektów (potencjalnego rozwiązania problemu)

Algorytmy z powrotami

Algorytmy z powrotami wykorzystujemy do rozwiązywania problemów, w których z określonego zbioru obiektów jest wybierana ich sekwencja spełniająca określone kryteria.
Algorytm z powrotami można interpretować jako przeszukiwanie w głąb drzewa reprezentującego tworzenie możliwych sekwencji obiektów (drzewa przestrzeni stanów):

- Zaczynamy od korzenia drzewa
- Dla każdego poddrzewa najpierw odwiedzamy jego korzeń, a następnie rekurencyjnie przeszukujemy wszystkich jego synów
- Odwiedzenie liścia drzewa oznacza wygenerowanie pełnej sekwencji obiektów (potencjalnego rozwiązania problemu)

Algorytmy z powrotami

Algorytmy z powrotami wykorzystujemy do rozwiązywania problemów, w których z określonego zbioru obiektów jest wybierana ich sekwencja spełniająca określone kryteria.
Algorytm z powrotami można interpretować jako przeszukiwanie w głąb drzewa reprezentującego tworzenie możliwych sekwencji obiektów (drzewa przestrzeni stanów):

- Zaczynamy od korzenia drzewa
- Dla każdego poddrzewa najpierw odwiedzamy jego korzeń, a następnie rekurencyjnie przeszukujemy wszystkich jego synów
- Odwiedzenie liścia drzewa oznacza wygenerowanie pełnej sekwencji obiektów (potencjalnego rozwiązania problemu)

Algorytmy z powrotami

Algorytmy z powrotami wykorzystujemy do rozwiązywania problemów, w których z określonego zbioru obiektów jest wybierana ich sekwencja spełniająca określone kryteria.
Algorytm z powrotami można interpretować jako przeszukiwanie w głąb drzewa reprezentującego tworzenie możliwych sekwencji obiektów (drzewa przestrzeni stanów):

- Zaczynamy od korzenia drzewa
- Dla każdego poddrzewa najpierw odwiedzamy jego korzeń, a następnie rekurencyjnie przeszukujemy wszystkich jego synów
- Odwiedzenie liścia drzewa oznacza wygenerowanie pełnej sekwencji obiektów (potencjalnego rozwiązania problemu)

Algorytmy z powrotami

Przykład 1: drzewo reprezentujące tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c".

- Korzeń drzewa przedstawia ciąg pusty
- Korzeń ma trzech synów o kluczach "a", "b" i "c" Reprezentują oni wybór odpowiedniej litery jako pierwszego elementu ciągu
- Każdy węzeł z pierwszego poziomu również ma trzech synów o kluczach "a", "b" i "c". Przedstawiają oni wybór odpowiedniej litery jako drugiego elementu ciągu
- Analogicznie, każdy węzeł z drugiego poziomu ma trzech synów odpowiadających za wybór trzeciego elementu ciągu
- Każda ścieżka od korzenia w dół do liścia odpowiada jednemu ciągowi długości 3

Algorytmy z powrotami

Przykład 1: drzewo reprezentujące tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c".

- Korzeń drzewa przedstawia ciąg pusty
- Korzeń ma trzech synów o kluczach "a", "b" i "c" Reprezentują oni wybór odpowiedniej litery jako pierwszego elementu ciągu
- Każdy węzeł z pierwszego poziomu również ma trzech synów o kluczach "a", "b" i "c". Przedstawiają oni wybór odpowiedniej litery jako drugiego elementu ciągu
- Analogicznie, każdy węzeł z drugiego poziomu ma trzech synów odpowiadających za wybór trzeciego elementu ciągu
- Każda ścieżka od korzenia w dół do liścia odpowiada jednemu ciągowi długości 3

Algorytmy z powrotami

Przykład 1: drzewo reprezentujące tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c".

- Korzeń drzewa przedstawia ciąg pusty
- Korzeń ma trzech synów o kluczach "a", "b" i "c" Reprezentują oni wybór odpowiedniej litery jako pierwszego elementu ciągu
- Każdy węzeł z pierwszego poziomu również ma trzech synów o kluczach "a", "b" i "c". Przedstawiają oni wybór odpowiedniej litery jako drugiego elementu ciągu
- Analogicznie, każdy węzeł z drugiego poziomu ma trzech synów odpowiadających za wybór trzeciego elementu ciągu
- Każda ścieżka od korzenia w dół do liścia odpowiada jednemu ciągowi długości 3

Algorytmy z powrotami

Przykład 1: drzewo reprezentujące tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c".

- Korzeń drzewa przedstawia ciąg pusty
- Korzeń ma trzech synów o kluczach "a", "b" i "c" Reprezentują oni wybór odpowiedniej litery jako pierwszego elementu ciągu
- Każdy węzeł z pierwszego poziomu również ma trzech synów o kluczach "a", "b" i "c". Przedstawiają oni wybór odpowiedniej litery jako drugiego elementu ciągu
- Analogicznie, każdy węzeł z drugiego poziomu ma trzech synów odpowiadających za wybór trzeciego elementu ciągu
- Każda ścieżka od korzenia w dół do liścia odpowiada jednemu ciągowi długości 3

Algorytmy z powrotami

Przykład 1: drzewo reprezentujące tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c".

- Korzeń drzewa przedstawia ciąg pusty
- Korzeń ma trzech synów o kluczach "a", "b" i "c" Reprezentują oni wybór odpowiedniej litery jako pierwszego elementu ciągu
- Każdy węzeł z pierwszego poziomu również ma trzech synów o kluczach "a", "b" i "c". Przedstawiają oni wybór odpowiedniej litery jako drugiego elementu ciągu
- Analogicznie, każdy węzeł z drugiego poziomu ma trzech synów odpowiadających za wybór trzeciego elementu ciągu
- Każda ścieżka od korzenia w dół do liścia odpowiada jednemu ciągowi długości 3

Algorytmy z powrotami

Przykład 1: drzewo reprezentujące tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c".

- Korzeń drzewa przedstawia ciąg pusty
- Korzeń ma trzech synów o kluczach "a", "b" i "c" Reprezentują oni wybór odpowiedniej litery jako pierwszego elementu ciągu
- Każdy węzeł z pierwszego poziomu również ma trzech synów o kluczach "a", "b" i "c". Przedstawiają oni wybór odpowiedniej litery jako drugiego elementu ciągu
- Analogicznie, każdy węzeł z drugiego poziomu ma trzech synów odpowiadających za wybór trzeciego elementu ciągu
- Każda ścieżka od korzenia w dół do liścia odpowiada jednemu ciągowi długości 3

Algorytmy z powrotami

- Przejście drzewa w głąb zaczyna się od przejścia jedną ścieżką od korzenia w dół tak głęboko, jak to jest możliwe. Po osiągnięciu ślepego zaułka powracamy w górę do najbliższego wierzchołka z nieodwiedzonymi synami i znów przechodzimy w głąb (kolejną ścieżką) tak daleko, jak to możliwe
- Jeśli generowane sekwencje mają spełniać pewne dodatkowe kryteria, w czasie odwiedzania pewnych węzłów może okazać się, że nie mogą one doprowadzić do poprawnego rozwiązania. Węzły takie nazywamy nieobiecującymi. Pozostałe węzły są obiecujące
- Jeśli stwierdzimy, że węzeł drzewa jest nieobiecujący, nie przeszukujemy jego synów, tylko od razu wracamy do jego ojca. Nazywamy to przycinaniem przestrzeni stanów
- Poddrzewo zawierające tylko odwiedzone węzły nazywamy przyciętym drzewem przestrzeni stanów

Algorytmy z powrotami

- Przejście drzewa w głąb zaczyna się od przejścia jedną ścieżką od korzenia w dół tak głęboko, jak to jest możliwe. Po osiągnięciu ślepego zaułka powracamy w górę do najbliższego wierzchołka z nieodwiedzonymi synami i znów przechodzimy w głąb (kolejną ścieżką) tak daleko, jak to możliwe
- Jeśli generowane sekwencje mają spełniać pewne dodatkowe kryteria, w czasie odwiedzania pewnych węzłów może okazać się, że nie mogą one doprowadzić do poprawnego rozwiązania. Węzły takie nazywamy nieobiecującymi. Pozostałe węzły są obiecujące
- Jeśli stwierdzimy, że węzeł drzewa jest nieobiecujący, nie przeszukujemy jego synów, tylko od razu wracamy do jego ojca. Nazywamy to przycinaniem przestrzeni stanów
- Poddrzewo zawierające tylko odwiedzone węzły nazywamy przyciętym drzewem przestrzeni stanów

Algorytmy z powrotami

- Przejście drzewa w głąb zaczyna się od przejścia jedną ścieżką od korzenia w dół tak głęboko, jak to jest możliwe. Po osiągnięciu ślepego zaułka powracamy w górę do najbliższego wierzchołka z nieodwiedzonymi synami i znów przechodzimy w głąb (kolejną ścieżką) tak daleko, jak to możliwe
- Jeśli generowane sekwencje mają spełniać pewne dodatkowe kryteria, w czasie odwiedzania pewnych węzłów może okazać się, że nie mogą one doprowadzić do poprawnego rozwiązania. Węzły takie nazywamy nieobiecującymi. Pozostałe węzły są obiecujące
- Jeśli stwierdzimy, że węzeł drzewa jest nieobiecujący, nie przeszukujemy jego synów, tylko od razu wracamy do jego ojca. Nazywamy to przycinaniem przestrzeni stanów
- Poddrzewo zawierające tylko odwiedzone węzły nazywamy przyciętym drzewem przestrzeni stanów

Algorytmy z powrotami

- Przejście drzewa w głąb zaczyna się od przejścia jedną ścieżką od korzenia w dół tak głęboko, jak to jest możliwe. Po osiągnięciu ślepego zaułka powracamy w górę do najbliższego wierzchołka z nieodwiedzonymi synami i znów przechodzimy w głąb (kolejną ścieżką) tak daleko, jak to możliwe
- Jeśli generowane sekwencje mają spełniać pewne dodatkowe kryteria, w czasie odwiedzania pewnych węzłów może okazać się, że nie mogą one doprowadzić do poprawnego rozwiązania. Węzły takie nazywamy nieobiecującymi. Pozostałe węzły są obiecujące
- Jeśli stwierdzimy, że węzeł drzewa jest nieobiecujący, nie przeszukujemy jego synów, tylko od razu wracamy do jego ojca. Nazywamy to przycinaniem przestrzeni stanów
- Poddrzewo zawierające tylko odwiedzone węzły nazywamy przyciętym drzewem przestrzeni stanów

Algorytmy z powrotami

Przykład 2: tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c", w których litery nie powtarzają się

- Pełne drzewo przestrzeni stanów wygląda tak samo, jak w poprzednim przykładzie
- Węzeł jest nieobiecujący, jeśli jego klucz jest równy kluczowi któregoś z jego przodków
- Wszystkie węzły z poziomu 1 są obiecujące
- Na poziomie 2 znajdują się trzy węzły nieobiecujące
- Drzewo zawiera również nieobiecujące liście, jednak nie umożliwiają one przycięcia przestrzeni stanów

Algorytmy z powrotami

Przykład 2: tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c", w których litery nie powtarzają się

- Pełne drzewo przestrzeni stanów wygląda tak samo, jak w poprzednim przykładzie
- Węzeł jest nieobiecujący, jeśli jego klucz jest równy kluczowi któregoś z jego przodków
- Wszystkie węzły z poziomu 1 są obiecujące
- Na poziomie 2 znajdują się trzy węzły nieobiecujące
- Drzewo zawiera również nieobiecujące liście, jednak nie umożliwiają one przycięcia przestrzeni stanów

Algorytmy z powrotami

Przykład 2: tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c", w których litery nie powtarzają się

- Pełne drzewo przestrzeni stanów wygląda tak samo, jak w poprzednim przykładzie
- Węzeł jest nieobiecujący, jeśli jego klucz jest równy kluczowi któregoś z jego przodków
- Wszystkie węzły z poziomu 1 są obiecujące
- Na poziomie 2 znajdują się trzy węzły nieobiecujące
- Drzewo zawiera również nieobiecujące liście, jednak nie umożliwiają one przycięcia przestrzeni stanów

Algorytmy z powrotami

Przykład 2: tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c", w których litery nie powtarzają się

- Pełne drzewo przestrzeni stanów wygląda tak samo, jak w poprzednim przykładzie
- Węzeł jest nieobiecujący, jeśli jego klucz jest równy kluczowi któregoś z jego przodków
- Wszystkie węzły z poziomu 1 są obiecujące
- Na poziomie 2 znajdują się trzy węzły nieobiecujące
- Drzewo zawiera również nieobiecujące liście, jednak nie umożliwiają one przycięcia przestrzeni stanów

Algorytmy z powrotami

Przykład 2: tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c", w których litery nie powtarzają się

- Pełne drzewo przestrzeni stanów wygląda tak samo, jak w poprzednim przykładzie
- Węzeł jest nieobiecujący, jeśli jego klucz jest równy kluczowi któregoś z jego przodków
- Wszystkie węzły z poziomu 1 są obiecujące
- Na poziomie 2 znajdują się trzy węzły nieobiecujące
- Drzewo zawiera również nieobiecujące liście, jednak nie umożliwiają one przycięcia przestrzeni stanów

Algorytmy z powrotami

Przykład 2: tworzenie wszystkich ciągów długości 3 składających się z liter "a", "b" i "c", w których litery nie powtarzają się

- Pełne drzewo przestrzeni stanów wygląda tak samo, jak w poprzednim przykładzie
- Węzeł jest nieobiecujący, jeśli jego klucz jest równy kluczowi któregoś z jego przodków
- Wszystkie węzły z poziomu 1 są obiecujące
- Na poziomie 2 znajdują się trzy węzły nieobiecujące
- Drzewo zawiera również nieobiecujące liście, jednak nie umożliwiają one przycięcia przestrzeni stanów

Algorytmy z powrotami

Ogólna struktura algorytmu z powrotami

```
procedure POWROTY(c)
 if OBIECUJĄCY(c)
  if ROZWIĄZANIE(c)
   WRITE(c)
  else for each v = ROZWINIĘCIE(c)
   POWROTY(v)
```

Sposób sprawdzania, czy częściowe rozwiązanie c jest obiecujące i czy jest rozwiązaniem całkowitym, oraz metoda tworzenia rozwinięcia v częściowego rozwiązania c zależą od konkretnego rozwiązywanego problemu

Algorytmy z powrotami

Ogólna struktura algorytmu z powrotami

```
procedure POWROTY(c)
 if OBIECUJĄCY(c)
  if ROZWIĄZANIE(c)
   WRITE(c)
  else for each v = ROZWINIĘCIE(c)
   POWROTY(v)
```

Sposób sprawdzania, czy częściowe rozwiązanie c jest obiecujące i czy jest rozwiązaniem całkowitym, oraz metoda tworzenia rozwinięcia v częściowego rozwiązania c zależą od konkretnego rozwiązywanego problemu

Algorytmy z powrotami

Ogólna struktura algorytmu z powrotami

```
procedure POWROTY(c)
 if OBIECUJĄCY(c)
  if ROZWIĄZANIE(c)
   WRITE(c)
  else for each v = ROZWINIĘCIE(c)
   POWROTY(v)
```

Sposób sprawdzania, czy częściowe rozwiązanie c jest obiecujące i czy jest rozwiązaniem całkowitym, oraz metoda tworzenia rozwinięcia v częściowego rozwiązania c zależą od konkretnego rozwiązywanego problemu

Algorytmy z powrotami

Problem n królowych

Znaleźć wszystkie sposoby ustawienia n królowych na szachownicy $n \times n$ w taki sposób, żeby się nie szachowały

Każda królowa musi znaleźć się w innym wierszu szachownicy. Mamy więc znaleźć sekwencję n numerów kolumn, w których należy umieścić królowe z kolejnych wierszy. Będziemy je zapisywać w tablicy $kol[1..n]$. Na i-tym poziomie drzewa przestrzeni stanów ustalamy wartość $kol[i]$, czyli numer kolumny dla i-tej królowej.

Algorytmy z powrotami

Problem n królowych

Znaleźć wszystkie sposoby ustawienia n królowych na szachownicy $n \times n$ w taki sposób, żeby się nie szachowały

Każda królowa musi znaleźć się w innym wierszu szachownicy. Mamy więc znaleźć sekwencję n numerów kolumn, w których należy umieścić królowe z kolejnych wierszy. Będziemy je zapisywać w tablicy $kol[1..n]$. Na i-tym poziomie drzewa przestrzeni stanów ustalamy wartość $kol[i]$, czyli numer kolumny dla i-tej królowej.

Algorytmy z powrotami

Problem n królowych
Znaleźć wszystkie sposoby ustawienia n królowych na szachownicy $n \times n$ w taki sposób, żeby się nie szachowały
Każda królowa musi znaleźć się w innym wierszu szachownicy. Mamy więc znaleźć sekwencję n numerów kolumn, w których należy umieścić królowe z kolejnych wierszy. Będziemy je zapisywać w tablicy $kol[1..n]$. Na i-tym poziomie drzewa przestrzeni stanów ustalamy wartość $kol[i]$, czyli numer kolumny dla i-tej królowej.

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich

Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych

Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.

Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich

Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych

Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.

Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich

Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych

Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.

Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich

Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych

Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.

Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich

Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych

Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.

Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich

Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych

Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.

Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

Królowe z wierszy i oraz k szachują się, jeśli:

- znajdują się w tej samej kolumnie: $kol[i] = kol[k]$
- znajdują się na jednej przekątnej: $kol[i] - kol[k] = i - k$ lub $kol[i] - kol[k] = k - i$

Aby dowiedzieć się, czy częściowe rozwiązanie jest obiecujące, wystarczy sprawdzić, czy ostatnia ustawiona królowa szachuje się z którąś z poprzednich
Rozwiązanie jest całkowite, jeśli zostały określone kolumny dla wszystkich n królowych
Częściowe rozwiązanie c jest określone przez tablicę kol oraz liczbę k ustawionych królowych. Przed rozpoczęciem algorytmu $k = 0$.
Rozwinięcie częściowego rozwiązania tworzymy ustalając numer kolumny dla królowej $k + 1$, czyli wpisując do $kol[k + 1]$ jedną z wartości $1, \ldots, n$

Algorytmy z powrotami

```
procedure POWROTY_KRÓLOWE(kol, k)
 if OBIECUJĄCY(kol, k)
  if k = n
   WRITE(kol)
  else for  i = 1 to n
   kol[k + 1] = i
   POWROTY_KRÓLOWE(kol, k + 1)
procedure OBIECUJĄCY(kol, k)
 for i = 1 to k - 1
  if kol[i] = kol[k] or |kol[i] − kol[k]| = |i − k|
   return false
  return true
POWROTY_KRÓLOWE(kol, 0)
```

Algorytmy z powrotami

```
procedure POWROTY_KRÓLOWE(kol, k)
 if OBIECUJĄCY(kol, k)
  if k = n
   WRITE(kol)
  else for  i = 1 to n
   kol[k + 1] = i
   POWROTY_KRÓLOWE(kol, k + 1)
procedure OBIECUJĄCY(kol, k)
 for i = 1 to k - 1
  if kol[i] = kol[k] or |kol[i] − kol[k]| = |i − k|
   return false
  return true
POWROTY_KRÓLOWE(kol, 0)
```

Algorytmy z powrotami

```
procedure POWROTY_KRÓLOWE(kol, k)
 if OBIECUJĄCY(kol, k)
  if k = n
   WRITE(kol)
  else for  i = 1 to n
   kol[k + 1] = i
   POWROTY_KRÓLOWE(kol, k + 1)
procedure OBIECUJĄCY(kol, k)
 for i = 1 to k - 1
  if kol[i] = kol[k] or |kol[i] − kol[k]| = |i − k|
   return false
  return true
POWROTY_KRÓLOWE(kol, 0)
```

Algorytmy z powrotami

Problem sumy podzbioru

Danych jest n liczb całkowitych dodatnich $m_1, \ldots, m_n oraz$ liczba naturalna M. Wyznaczyć wszystkie kombinacje liczb ze zbioru $\{m_1, \ldots, m_n\}$ dające w sumie M.

Każda z liczb $m_1, \ldots, m_n$ może należeć lub nie należeć do utworzonego podzbioru. Podzbiór możemy więc zakodować jako ciąg binarny długości n (jedynka oznacza zaliczenie liczby do zbioru, a zero - odrzucenie jej). Będziemy go zapisywać w tablicy $x[1..n]$. Na i-tym poziomie drzewa przestrzeni stanów decydujemy, czy liczba m_i należy do podzbioru, ustalając $x[i] = 1$ lub $x[i] = 0$.

Algorytmy z powrotami

Problem sumy podzbioru
Danych jest n liczb całkowitych dodatnich $m_1, \ldots, m_n oraz$ liczba naturalna M. Wyznaczyć wszystkie kombinacje liczb ze zbioru $\{m_1, \ldots, m_n\}$ dające w sumie M.

Każda z liczb $m_1, \ldots, m_n$ może należeć lub nie należeć do utworzonego podzbioru. Podzbiór możemy więc zakodować jako ciąg binarny długości n (jedynka oznacza zaliczenie liczby do zbioru, a zero - odrzucenie jej). Będziemy go zapisywać w tablicy $x[1..n]$. Na i-tym poziomie drzewa przestrzeni stanów decydujemy, czy liczba m_i należy do podzbioru, ustalając $x[i] = 1$ lub $x[i] = 0$.

Algorytmy z powrotami

Problem sumy podzbioru
Danych jest n liczb całkowitych dodatnich $m_1, \ldots, m_n oraz$ liczba naturalna M. Wyznaczyć wszystkie kombinacje liczb ze zbioru $\{m_1, \ldots, m_n\}$ dające w sumie M.
Każda z liczb $m_1, \ldots, m_n$ może należeć lub nie należeć do utworzonego podzbioru. Podzbiór możemy więc zakodować jako ciąg binarny długości n (jedynka oznacza zaliczenie liczby do zbioru, a zero - odrzucenie jej). Będziemy go zapisywać w tablicy $x[1..n]$. Na i-tym poziomie drzewa przestrzeni stanów decydujemy, czy liczba m_i należy do podzbioru, ustalając $x[i] = 1$ lub $x[i] = 0$.

Algorytmy z powrotami

Przyjmijmy, że przed rozpoczęciem działania algorytmu liczby mi zostały posortowane niemalejąco, $m_1 \leq m_2 \leq \cdots \leq m_n$. Zatem m_{i+1} jest najmniejszą liczbą pozostałą do wyboru poniżej poziomu i.

Niech waga będzie sumą liczb wybranych do podzbioru w częściowym rozwiązaniu na poziomie i. Jeśli $waga = M$, to węzeł zawiera poprawne rozwiązanie. W takim przypadku wypisujemy je i wracamy do ojca bieżącego węzła. Zatem algorytm z powrotami dla problemu sumy podzbioru może znaleźć rozwiązanie przed osiągnięciem liścia drzewa przestrzeni stanów.

Jeśli $waga \neq M$ oraz $waga + m_{i+1} > M$, to węzeł jest nieobiecujący.

Algorytmy z powrotami

Przyjmijmy, że przed rozpoczęciem działania algorytmu liczby mi zostały posortowane niemalejąco, $m_1 \leq m_2 \leq \cdots \leq m_n$. Zatem m_{i+1} jest najmniejszą liczbą pozostałą do wyboru poniżej poziomu i. Niech waga będzie sumą liczb wybranych do podzbioru w częściowym rozwiązaniu na poziomie i. Jeśli $waga = M$, to węzeł zawiera poprawne rozwiązanie. W takim przypadku wypisujemy je i wracamy do ojca bieżącego węzła. Zatem algorytm z powrotami dla problemu sumy podzbioru może znaleźć rozwiązanie przed osiągnięciem liścia drzewa przestrzeni stanów.

Jeśli $waga \neq M$ oraz $waga + m_{i+1} > M$, to węzeł jest nieobiecujący.

Algorytmy z powrotami

Przyjmijmy, że przed rozpoczęciem działania algorytmu liczby mi zostały posortowane niemalejąco, $m_1 \leq m_2 \leq \cdots \leq m_n$. Zatem m_{i+1} jest najmniejszą liczbą pozostałą do wyboru poniżej poziomu i.
Niech waga będzie sumą liczb wybranych do podzbioru w częściowym rozwiązaniu na poziomie i. Jeśli $waga = M$, to węzeł zawiera poprawne rozwiązanie. W takim przypadku wypisujemy je i wracamy do ojca bieżącego węzła. Zatem algorytm z powrotami dla problemu sumy podzbioru może znaleźć rozwiązanie przed osiągnięciem liścia drzewa przestrzeni stanów.
Jeśli $waga \neq M$ oraz $waga + m_{i+1} > M$, to węzeł jest nieobiecujący.

Algorytmy z powrotami

Problem plecakowy

Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku. Jest to problem optymalizacyjny.
Inne sformułowanie: znaleźć ciąg $(x_1, \ldots, x_n) \in \{0, 1\}^n$, dla którego

$$\sum_{i=1}^{n} p_i x_i$$

jest największa, przy ograniczeniu

$$\sum_{i=1}^{n} m_i x_i \leq M$$

.

Algorytmy z powrotami

Problem plecakowy

Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku. Jest to problem optymalizacyjny.

Inne sformułowanie: znaleźć ciąg $(x_1, \ldots, x_n) \in \{0, 1\}^n$, dla którego

$$\sum_{i=1}^{n} p_i x_i$$

jest największa, przy ograniczeniu

$$\sum_{i=1}^{n} m_i x_i \leq M$$

.

Algorytmy z powrotami

Problem plecakowy
Mamy plecak, w którym można przenieść przedmioty o łącznej wadze nieprzekraczającej M. Danych jest n przedmiotów $e_1, \ldots, e_n$, przy czym przedmiot e_i ma wartość p_i i wagę m_i. Znaleźć jak najcenniejszy zbiór przedmiotów, który można przenieść w plecaku. Jest to problem optymalizacyjny.
Inne sformułowanie: znaleźć ciąg $(x_1, \ldots, x_n) \in \{0, 1\}^n$, dla którego

$$\sum_{i=1}^{n} p_i x_i$$

jest największa, przy ograniczeniu

$$\sum_{i=1}^{n} m_i x_i \leq M$$

.

Algorytmy z powrotami

Rozwiązanie naiwne: przeglądamy wszystkie podzbiory zbioru przedmiotów, odrzucamy te, których waga jest za duża, a z pozostałych wybieramy ten najcenniejszy. Złożoność: $\Omega(2^n)$.

Rozwiązanie za pomocą algorytmu z powrotami: podobnie jak dla sumy podzbiorów, na i-tym poziomie drzewa podejmujemy decyzję o wybraniu lub odrzuceniu i-tego przedmiotu, ustalając $x_i = 1$ lub $x_i = 0$

Ponieważ rozwiązujemy problem optymalizacyjny, musimy zapamiętywać najlepsze znalezione dotychczas rozwiązanie. W każdym obiecującym węźle sprawdzamy łączną wartość wybranych przedmiotów i jeśli jest ona większa od wartości zapamiętanego rozwiązania, to zapamiętujemy bieżące rozwiązanie jako najlepsze.

Algorytmy z powrotami

Rozwiązanie naiwne: przeglądamy wszystkie podzbiory zbioru przedmiotów, odrzucamy te, których waga jest za duża, a z pozostałych wybieramy ten najcenniejszy. Złożoność: $\Omega(2^n)$.
Rozwiązanie za pomocą algorytmu z powrotami: podobnie jak dla sumy podzbiorów, na i-tym poziomie drzewa podejmujemy decyzję o wybraniu lub odrzuceniu i-tego przedmiotu, ustalając $x_i = 1$ lub $x_i = 0$

Ponieważ rozwiązujemy problem optymalizacyjny, musimy zapamiętywać najlepsze znalezione dotychczas rozwiązanie. W każdym obiecującym węźle sprawdzamy łączną wartość wybranych przedmiotów i jeśli jest ona większa od wartości zapamiętanego rozwiązania, to zapamiętujemy bieżące rozwiązanie jako najlepsze.

Algorytmy z powrotami

Rozwiązanie naiwne: przeglądamy wszystkie podzbiory zbioru przedmiotów, odrzucamy te, których waga jest za duża, a z pozostałych wybieramy ten najcenniejszy. Złożoność: $\Omega(2^n)$.
Rozwiązanie za pomocą algorytmu z powrotami: podobnie jak dla sumy podzbiorów, na i-tym poziomie drzewa podejmujemy decyzję o wybraniu lub odrzuceniu i-tego przedmiotu, ustalając $x_i = 1$ lub $x_i = 0$
Ponieważ rozwiązujemy problem optymalizacyjny, musimy zapamiętywać najlepsze znalezione dotychczas rozwiązanie. W każdym obiecującym węźle sprawdzamy łączną wartość wybranych przedmiotów i jeśli jest ona większa od wartości zapamiętanego rozwiązania, to zapamiętujemy bieżące rozwiązanie jako najlepsze.

Algorytmy z powrotami

Niech zmienna *waga* oznacza łączną wagę, a zmienna *profit* - łączną wartość przedmiotów należących do częściowego rozwiązania problemu.

Rozwiązanie częściowe jest nieobiecujące, jeśli $waga > M$. Jeśli $waga = M$, to mamy poprawne rozwiązanie, do którego nie można już dołączyć żadnego przedmiotu. W obu przypadkach przycinamy drzewo przestrzeni stanów, wracając na poprzedni poziom.

Do odrzucania rozwiązań częściowych możemy wykorzystać nie tylko wagę przedmiotów, ale także stosunek ich wartości do wartości najlepszego znalezionego dotychczas rozwiązania.

Zakładamy teraz, że przedmioty zostały posortowane nierosnąco według wartości $\frac{p_i}{m_i}$.

Algorytmy z powrotami

Niech zmienna *waga* oznacza łączną wagę, a zmienna *profit* - łączną wartość przedmiotów należących do częściowego rozwiązania problemu.
Rozwiązanie częściowe jest nieobiecujące, jeśli $waga > M$. Jeśli $waga = M$, to mamy poprawne rozwiązanie, do którego nie można już dołączyć żadnego przedmiotu. W obu przypadkach przycinamy drzewo przestrzeni stanów, wracając na poprzedni poziom.
Do odrzucania rozwiązań częściowych możemy wykorzystać nie tylko wagę przedmiotów, ale także stosunek ich wartości do wartości najlepszego znalezionego dotychczas rozwiązania.
Zakładamy teraz, że przedmioty zostały posortowane nierosnąco według wartości $\frac{p_i}{m_i}$.

Algorytmy z powrotami

Niech zmienna *waga* oznacza łączną wagę, a zmienna *profit* - łączną wartość przedmiotów należących do częściowego rozwiązania problemu.
Rozwiązanie częściowe jest nieobiecujące, jeśli $waga > M$. Jeśli $waga = M$, to mamy poprawne rozwiązanie, do którego nie można już dołączyć żadnego przedmiotu. W obu przypadkach przycinamy drzewo przestrzeni stanów, wracając na poprzedni poziom.
Do odrzucania rozwiązań częściowych możemy wykorzystać nie tylko wagę przedmiotów, ale także stosunek ich wartości do wartości najlepszego znalezionego dotychczas rozwiązania.
Zakładamy teraz, że przedmioty zostały posortowane nierosnąco według wartości $\frac{p_i}{m_i}$.

Algorytmy z powrotami

- Dla bieżącego węzła na poziomie i sprawdzamy, ile kolejnych przedmiotów (od numeru $i+1$) możemy dołączyć, nie przekraczając ograniczenia wagi M
- Niech przedmiot k będzie tym, który powoduje przekroczenie ograniczenia wagi. Definiujemy

$$calkowitawaga = waga + \sum_{j=i+1}^{k-1} m_j$$

- Wówczas

$$granica = (profit + \sum_{j=i+1}^{k-1} p_j) + (M - calkowitawaga)\frac{p_k}{m_k}$$

jest górnym ograniczeniem na wartość rozwinięcia bieżącego rozwiązania.

- Zatem, jeśli *granica* jest nie większa niż wartość najlepszego znalezionego dotychczas rozwiązania, to bieżący węzeł jest

Algorytmy z powrotami

- Dla bieżącego węzła na poziomie i sprawdzamy, ile kolejnych przedmiotów (od numeru $i+1$) możemy dołączyć, nie przekraczając ograniczenia wagi M
- Niech przedmiot k będzie tym, który powoduje przekroczenie ograniczenia wagi. Definiujemy

$$calkowitawaga = waga + \sum_{j=i+1}^{k-1} m_j$$

- Wówczas

$$granica = (profit + \sum_{j=i+1}^{k-1} p_j) + (M - calkowitawaga)\frac{p_k}{m_k}$$

jest górnym ograniczeniem na wartość rozwinięcia bieżącego rozwiązania.

- Zatem, jeśli *granica* jest nie większa niż wartość najlepszego znalezionego dotychczas rozwiązania, to bieżący węzeł jest

Algorytmy z powrotami

- Dla bieżącego węzła na poziomie i sprawdzamy, ile kolejnych przedmiotów (od numeru $i+1$) możemy dołączyć, nie przekraczając ograniczenia wagi M
- Niech przedmiot k będzie tym, który powoduje przekroczenie ograniczenia wagi. Definiujemy

$$calkowitawaga = waga + \sum_{j=i+1}^{k-1} m_j$$

- Wówczas

$$granica = (profit + \sum_{j=i+1}^{k-1} p_j) + (M - calkowitawaga)\frac{p_k}{m_k}$$

  jest górnym ograniczeniem na wartość rozwinięcia bieżącego rozwiązania.
- Zatem, jeśli *granica* jest nie większa niż wartość najlepszego znalezionego dotychczas rozwiązania, to bieżący węzeł jest

Algorytmy z powrotami

- Dla bieżącego węzła na poziomie i sprawdzamy, ile kolejnych przedmiotów (od numeru $i+1$) możemy dołączyć, nie przekraczając ograniczenia wagi M
- Niech przedmiot k będzie tym, który powoduje przekroczenie ograniczenia wagi. Definiujemy

$$calkowitawaga = waga + \sum_{j=i+1}^{k-1} m_j$$

- Wówczas

$$granica = (profit + \sum_{j=i+1}^{k-1} p_j) + (M - calkowitawaga)\frac{p_k}{m_k}$$

  jest górnym ograniczeniem na wartość rozwinięcia bieżącego rozwiązania.
- Zatem, jeśli *granica* jest nie większa niż wartość najlepszego znalezionego dotychczas rozwiązania, to bieżący węzeł jest

Algorytmy z powrotami

Problem plecakowy – przykład

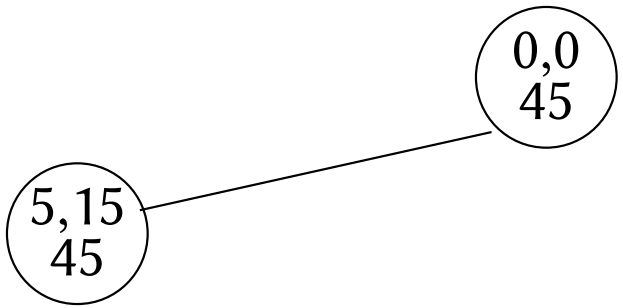
- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

- najlepsze znane rozwiązanie: $waga = 0$, $profit = 0$, []

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

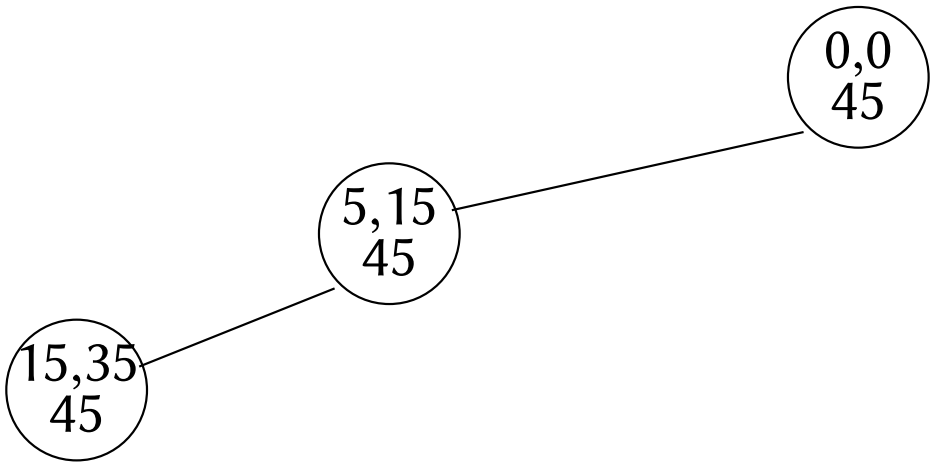


- najlepsze znane rozwiązanie: *waga* $= 5$, *profit* $= 15$, $[1]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

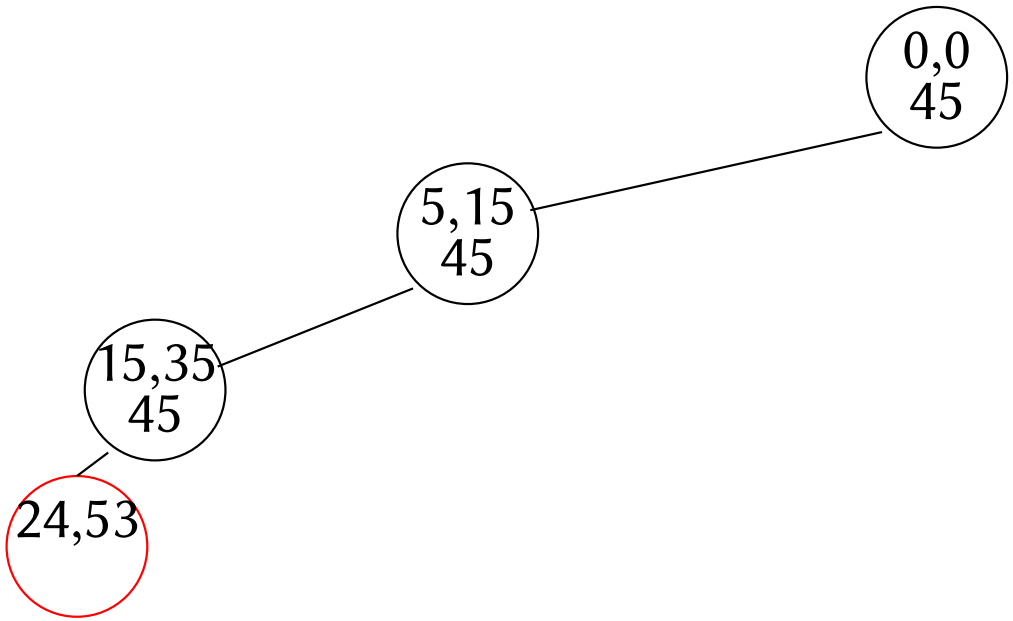


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

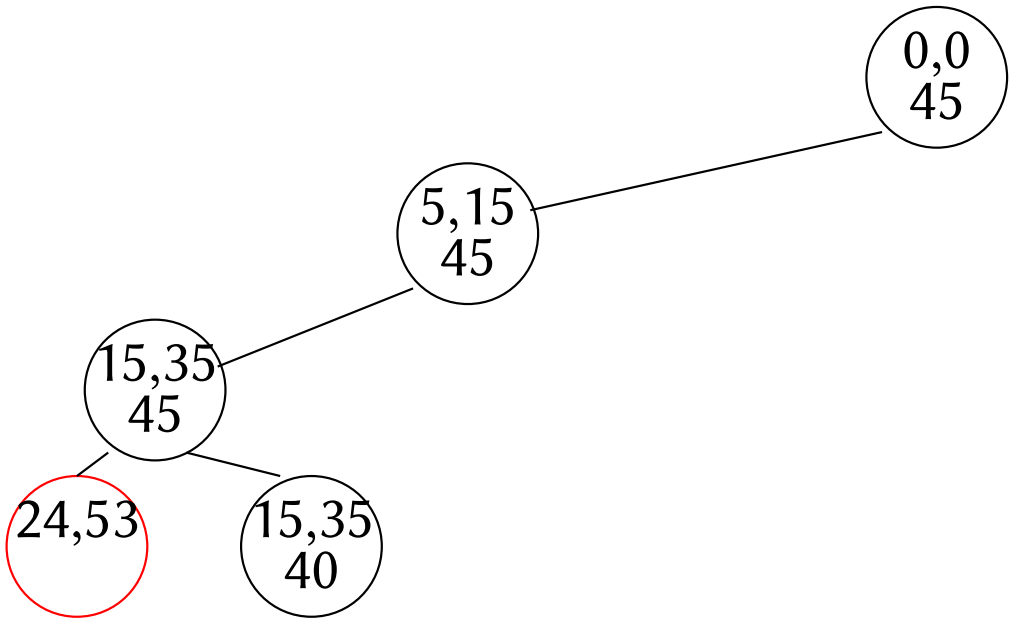


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

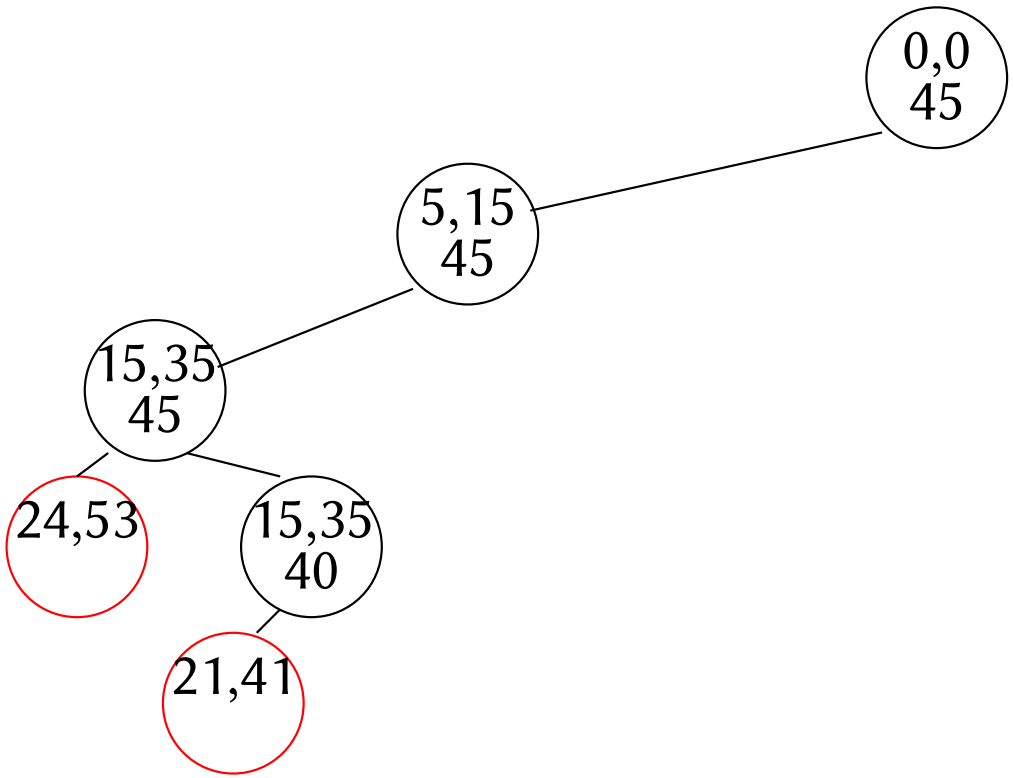


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

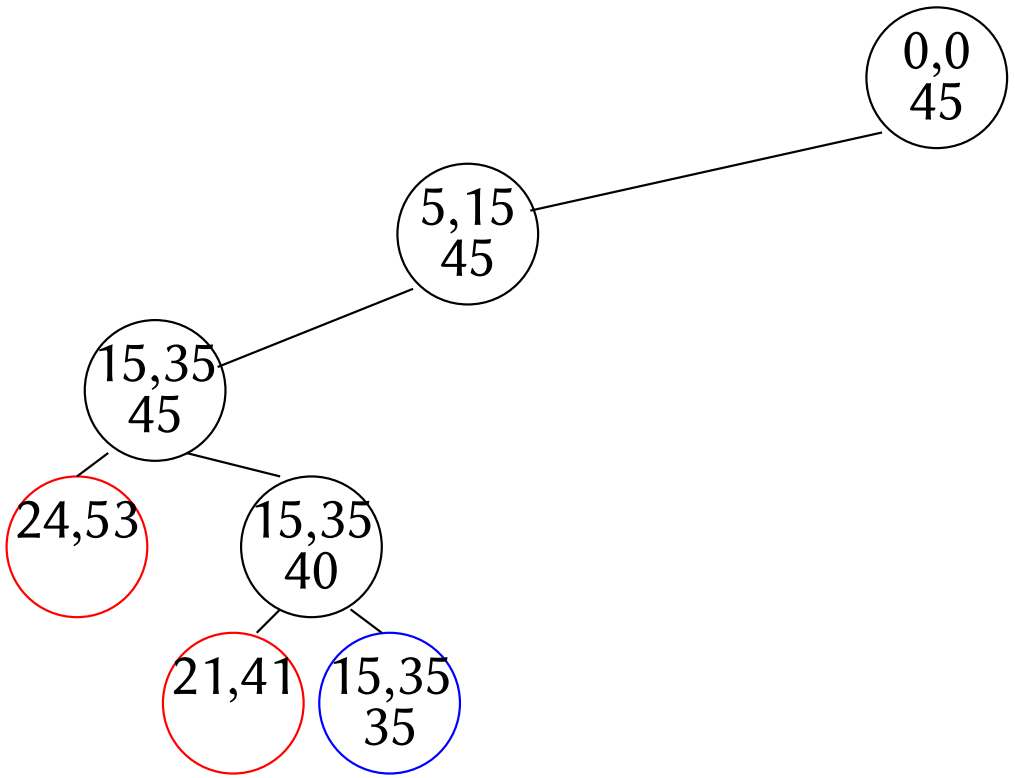


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

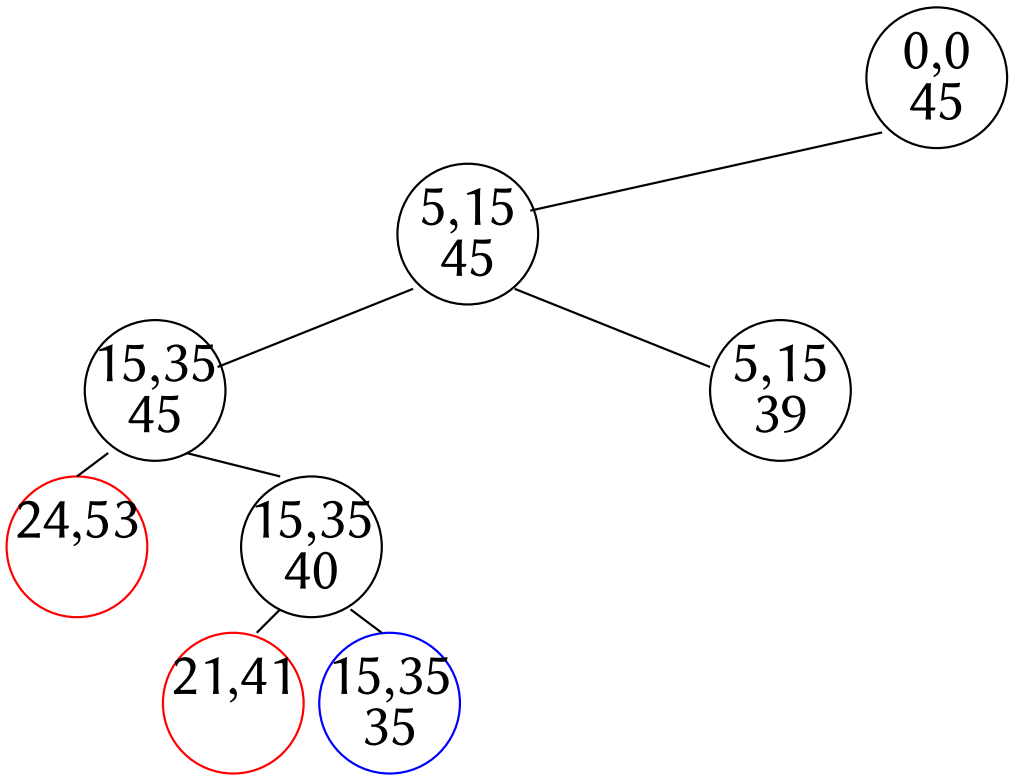


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

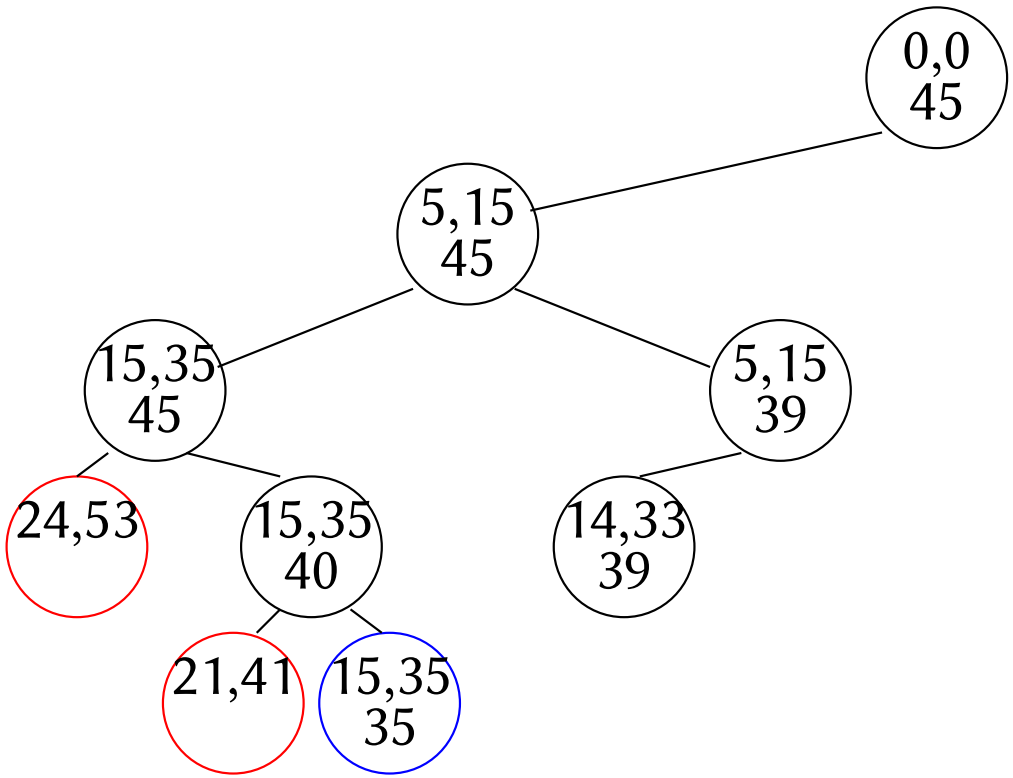


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

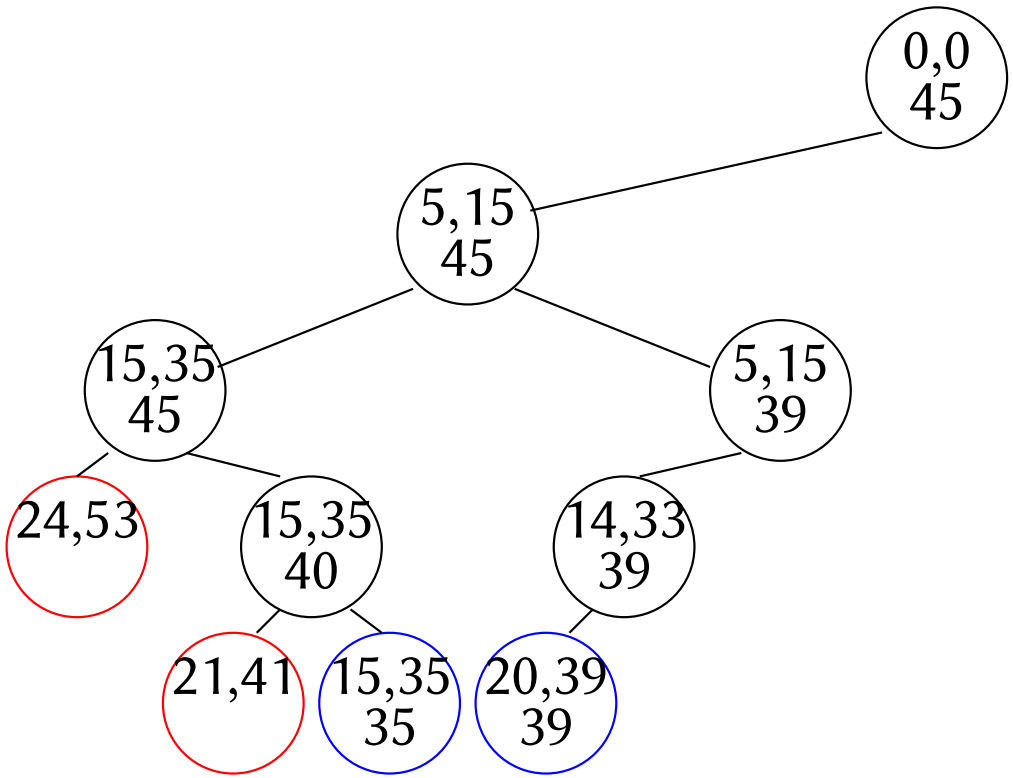


- najlepsze znane rozwiązanie: *waga* $= 15$, *profit* $= 35$, $[1, 2]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

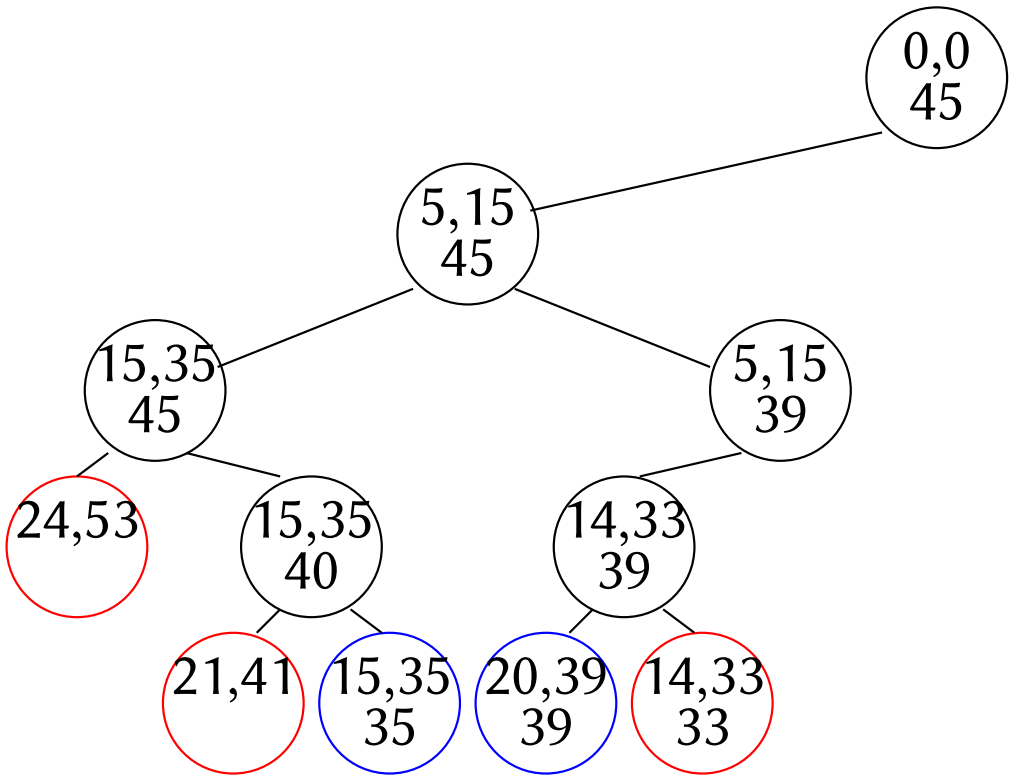


- najlepsze znane rozwiązanie: *waga* $= 20$, *profit* $= 39$, $[1, 3, 4]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

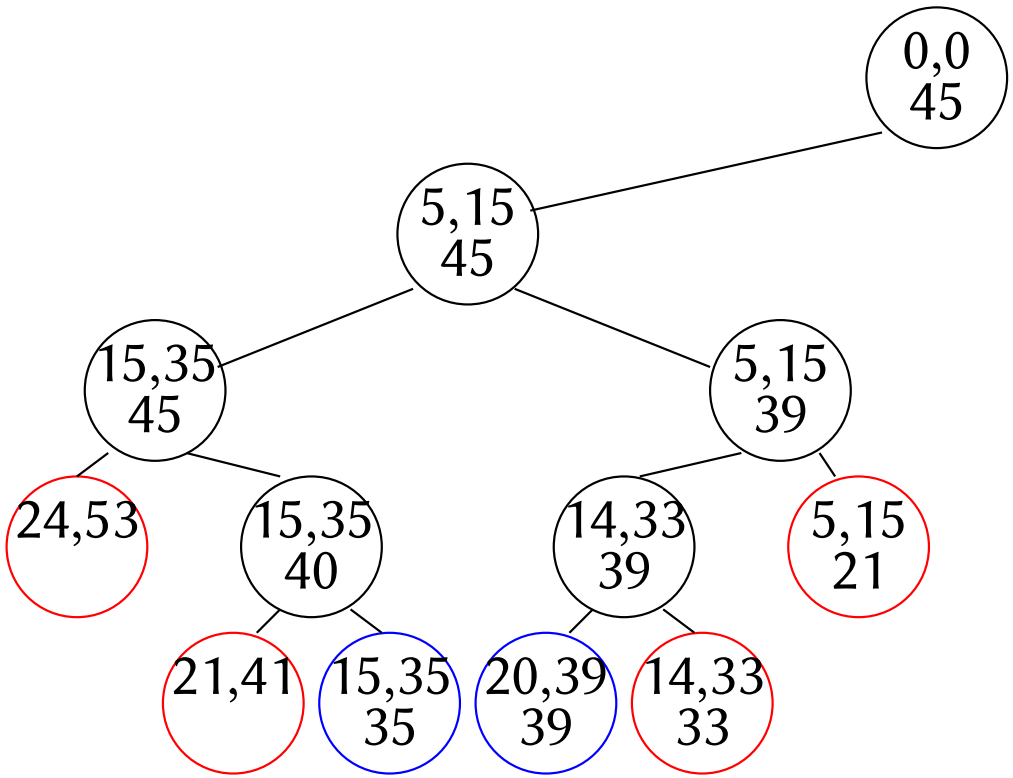


- najlepsze znane rozwiązanie: *waga* $= 20$, *profit* $= 39$, $[1, 3, 4]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*

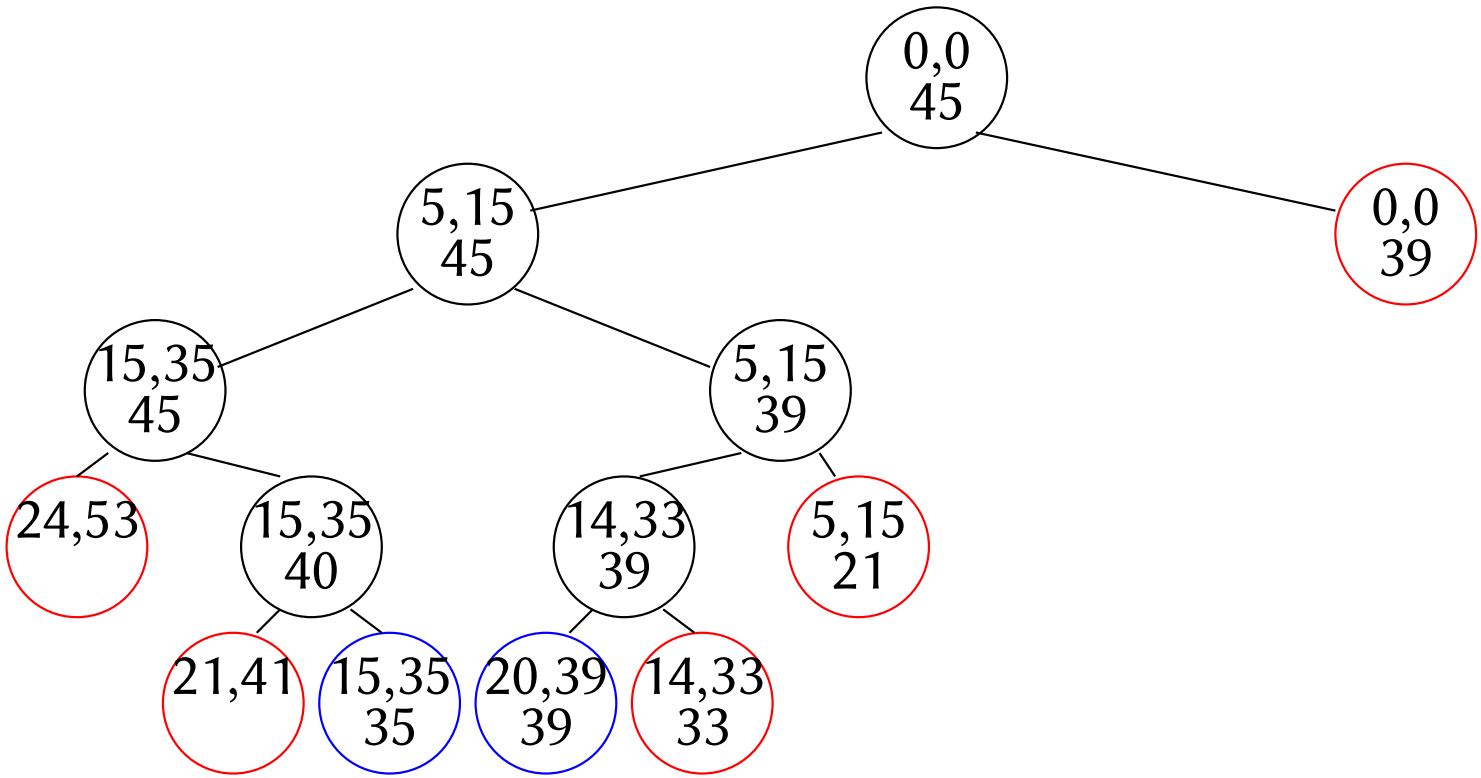


- najlepsze znane rozwiązanie: *waga* $= 20$, *profit* $= 39$, $[1, 3, 4]$

Algorytmy z powrotami

Problem plecakowy – przykład

- $n = 4$, $M = 20$, $p = [15, 20, 18, 6]$, $m = [5, 10, 9, 6]$
- w węzłach zapisujemy kolejno wartości *waga*, *profit*, *granica*



- najlepsze znane rozwiązanie: *waga* $= 20$, *profit* $= 39$, $[1, 3, 4]$