Metoda podziału i ograniczeń

Mateusz Łyczek

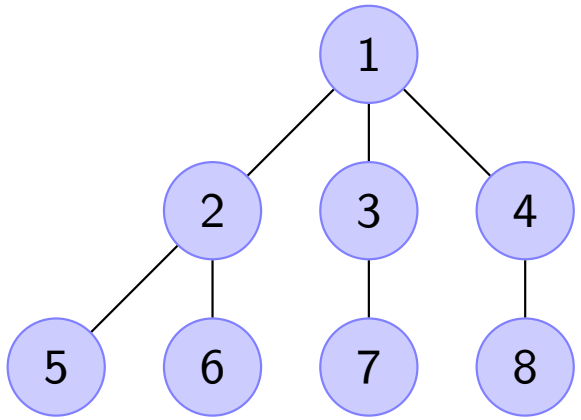
Instytut Informatyki Uniwersytetu Wrocławskiego

Seminarium: Algorytmy heurystyczne

16 marca 2011 r.

Drzewo przestrzeni stanów

- reprezentuje przestrzeń rozwiązań
- każdy wierzchołek reprezentuje stan algorytmu
- algorytm rozpoczyna w korzeniu drzewa i przechodząc drzewo konstruuje rozwiązanie



Wada: wykładniczy rozmiar drzewa

Metoda podziału i ograniczeń

- dobrze byłoby więc "przyciąć" drzewo, żeby nie przeglądać go całego
- wyliczamy **ograniczenia** na wartość rozwiązania, które możemy uzyskać z potomków danego węzła
- pamiętamy najlepsze dotychczas znalezione rozwiązanie
- możemy dzięki temu **dzielić** (przycinać) drzewo przestrzeni poszukiwań i przeglądać tylko obiecujące obszary

Granice oraz węzły obiecujące i nieobiecujące

Granica

Liczba ta jest ograniczeniem wartości rozwiązania jakie można uzyskać dzięki rozwinięciu danego węzła (przeglądaniu jego potomków)

Węzeł obiecujący

Węzeł jest **obiecujący** jeśli jego granica jest *lepsza* niż wartość najlepszego znalezionego do tej pory rozwiązania

Węzeł nieobiecujący

Wartość jego granicy jest *gorsza* od wartości najlepszego znalezionego do tej pory rozwiązania

Składowe metody podziału i ograniczeń

Algorytm korzystający z metody podziału i ograniczeń musi zapewnić dwa elementy:

1. **funkcję obliczającą granicę**, która potrafi dla rozwiązywanego problemu wyznaczyć granicę najlepszego rozwiązania jakie można otrzymać z poddrzewa danego węzła
2. **strategię odwiedzania wierzchołków** w drzewie przestrzeni stanów, gdyż ma ona istotny wpływ na szybkość znalezienia rozwiązania

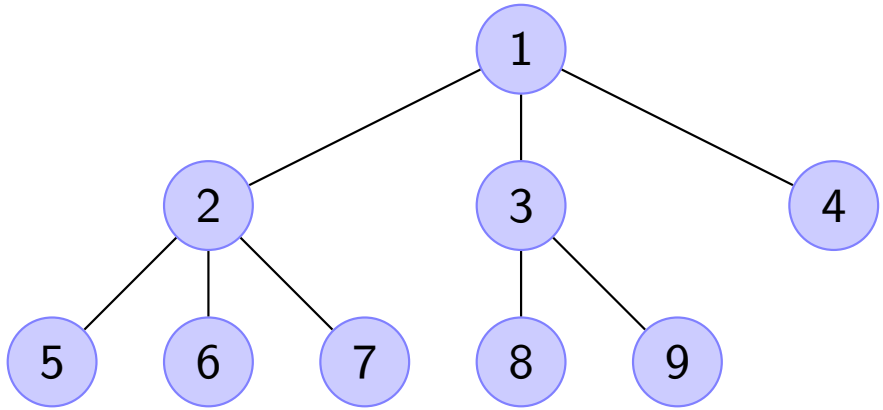
Dwie strategie

Zobaczymy dwie strategie przeglądania wierzchołków:

- przeszukiwanie wszerz
- przeszukiwanie typu najpierw najlepszy

Przeszukiwanie wszerz

initialize(Q)
$v \leftarrow$ korzeń drzewa
odwiedź v
enqueue(Q, v)
while !*empty*(Q) **do**
    $v \leftarrow$ *dequeue*(Q)
    for all dzieci u wierzchołka v **do**
        odwiedź u
        enqueue(Q, u)
    end for
end while



Problem plecakowy 0-1

Definicja

- mamy n przedmiotów o podanej wadze i wartości
- waga i wartość są liczbami całkowitymi dodatnimi
- mamy podaną dodatnią liczbę W

Problem: znaleźć zbiór przedmiotów o maksymalnej wartości sumarycznej których suma wag nie przekracza W

Przykład

$n = 4$
$W = 16$

i	p_i	w_i	$\frac{p_i}{w_i}$
1	40 zł	2	20 zł
2	30 zł	5	6 zł
3	50 zł	10	5 zł
4	10 zł	5	2 zł

Przykład - Problem plecakowy 0-1

- przechowujemy zmienną *maxprofit* pamiętającą najwyższy zysk do tej pory znaleziony
- dla każdego węzła przechowujemy wartości
  - **weight** - waga całkowita przedmiotów w węźle
  - **profit** - całkowita wartość przedmiotów w węźle
  - **totweight** - waga przedmiotów po dodaniu następnych elementów (przydatne przy obliczaniu granicy)
  - **bound** - górna granica zysku jaką możemy osiągnąć poniżej tego węzła

Problem plecakowy 0-1 - Obliczanie granicy

1. inicjujemy zmienne *totweight* $\leftarrow$ *weight* oraz *bound* $\leftarrow$ *profit*
2. zachłannie zbieramy przedmioty dodając ich wagi i wartości odpowiednio do zmiennych *totweight* i *bound*
3. robimy to do momentu aż natrafimy na przedmiot, który spowoduje że *totweight* przekroczy W
4. zabieramy tylko część tego elementu, żeby nie przekroczyć W
5. dodajemy zysk odpowiednio przeskalowany do zmiennej *bound*

Problem plecakowy 0-1 - Obliczanie granicy

Jeżeli węzeł przepełniający znajduje się na poziomie i, a węzeł przepełniający znajduje się na poziomie k to:

$$totweight = weight + \sum_{j=i+1}^{k-1} w_j$$

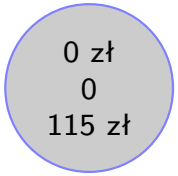
$$bound = \left(profit + \sum_{j=i+1}^{k-1} \right) + (W - totweight) \times \frac{p_k}{w_k}$$

Algorytm

initialize(Q)
$v \leftarrow$ korzeń drzewa (*drzewo w aplikacji istnieje niejawnie*)
enqueue(Q, v)
best $\leftarrow$ *value*(v)
while !*empty*(Q) **do**
  $v \leftarrow$ *dequeue*(Q)
  for all każde dziecko u węzła v **do**
    if *value*(u) lepsza od *best* **then**
      best $\leftarrow$ *value*(u)
    end if
    if *bound*(u) jest lepsza od *best* **then**
      enqueue(Q, u)
    end if
  end for
end while

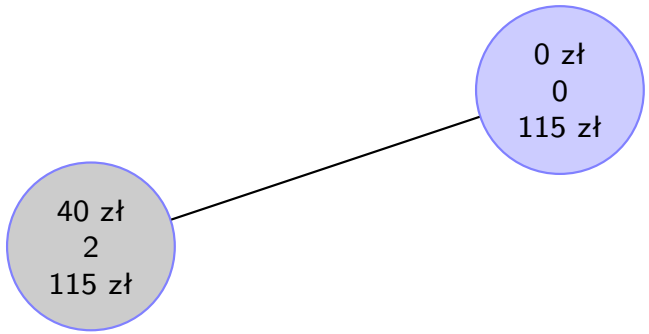
Przeszukiwanie typu najpierw najlepszy

- Strategia przeszukiwania wszerz nie ma żadnych dodatkowych zalet w porównaniu z przeszukiwaniem w głąb (w metodzie z powrotami)
- możemy po odwiedzeniu wszystkich bezpośrednich dzieci danego węzła przeanalizować wszystkie nierozwinięte jeszcze obiecujące węzły i wybrać najlepszy z nich jako następny do rozwijania



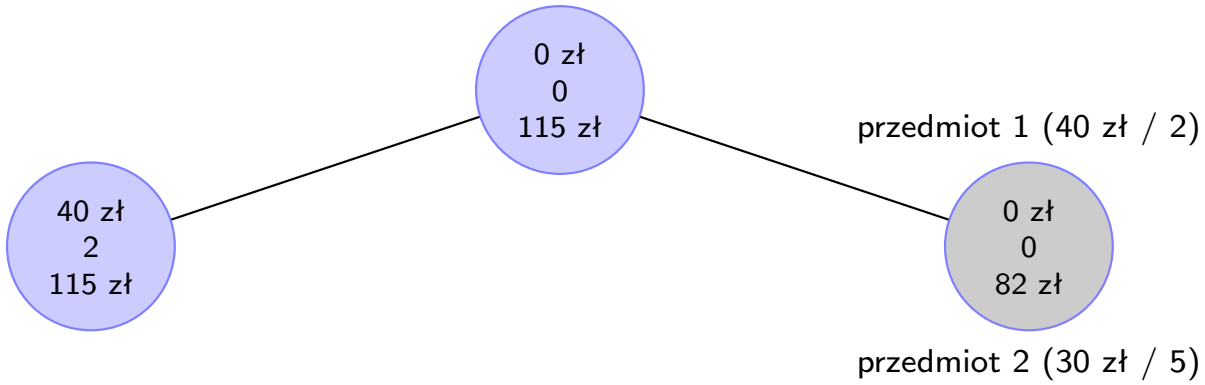
przedmiot 1 (40 zł / 2)

$maxprofit = 0$

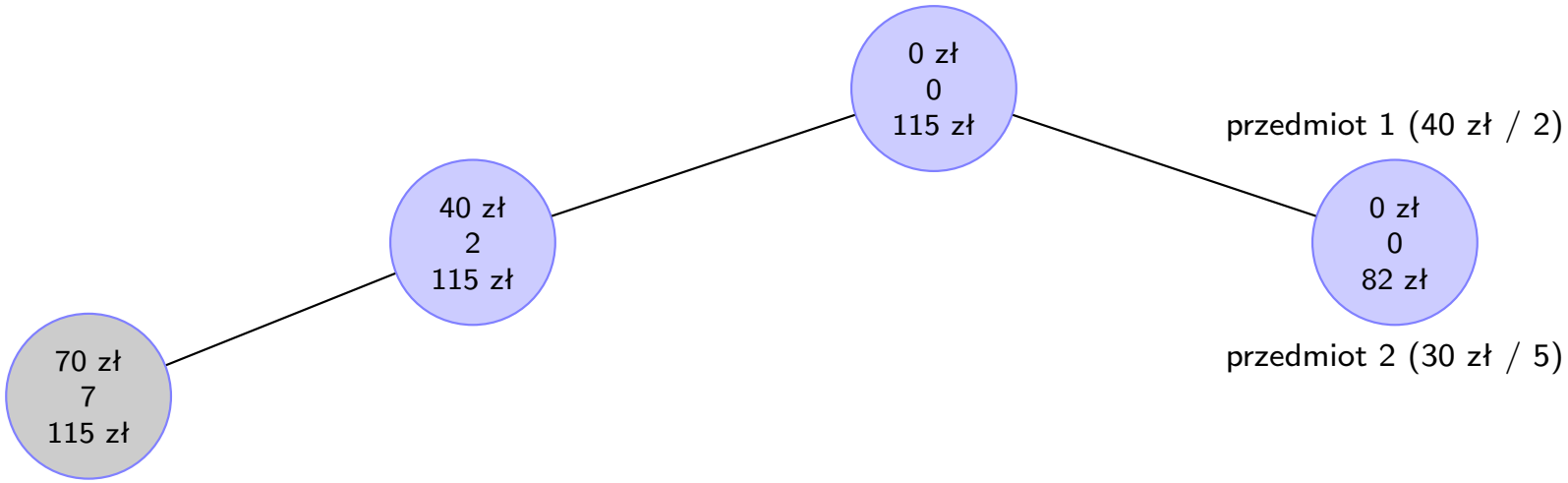


przedmiot 1 (40 zł / 2)

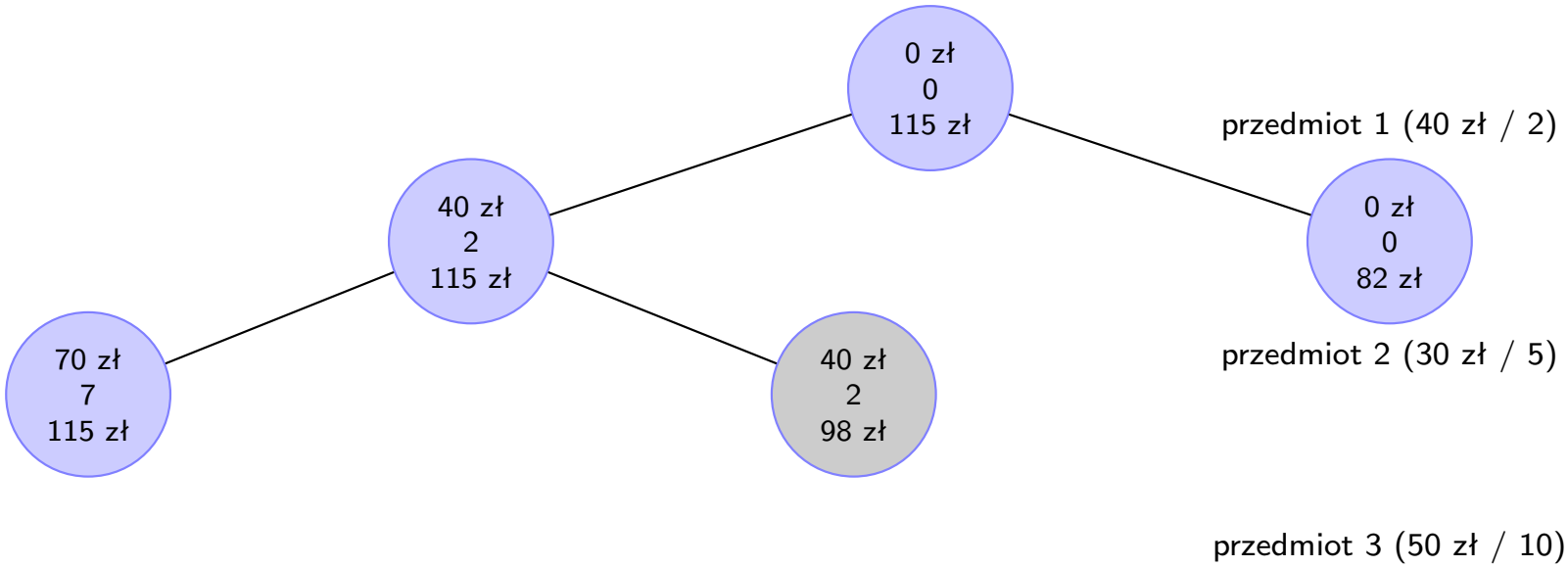
$maxprofit = 40$



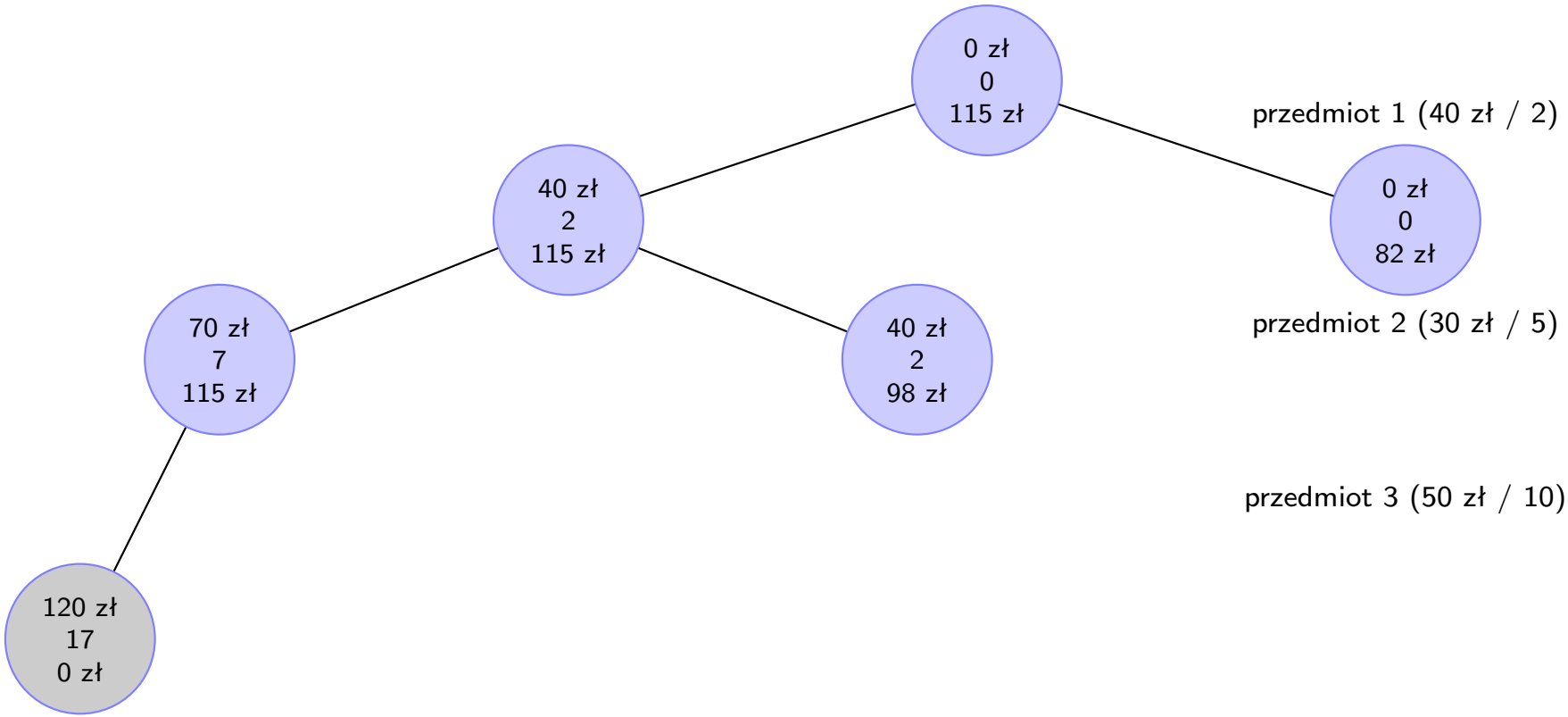
$maxprofit = 40$



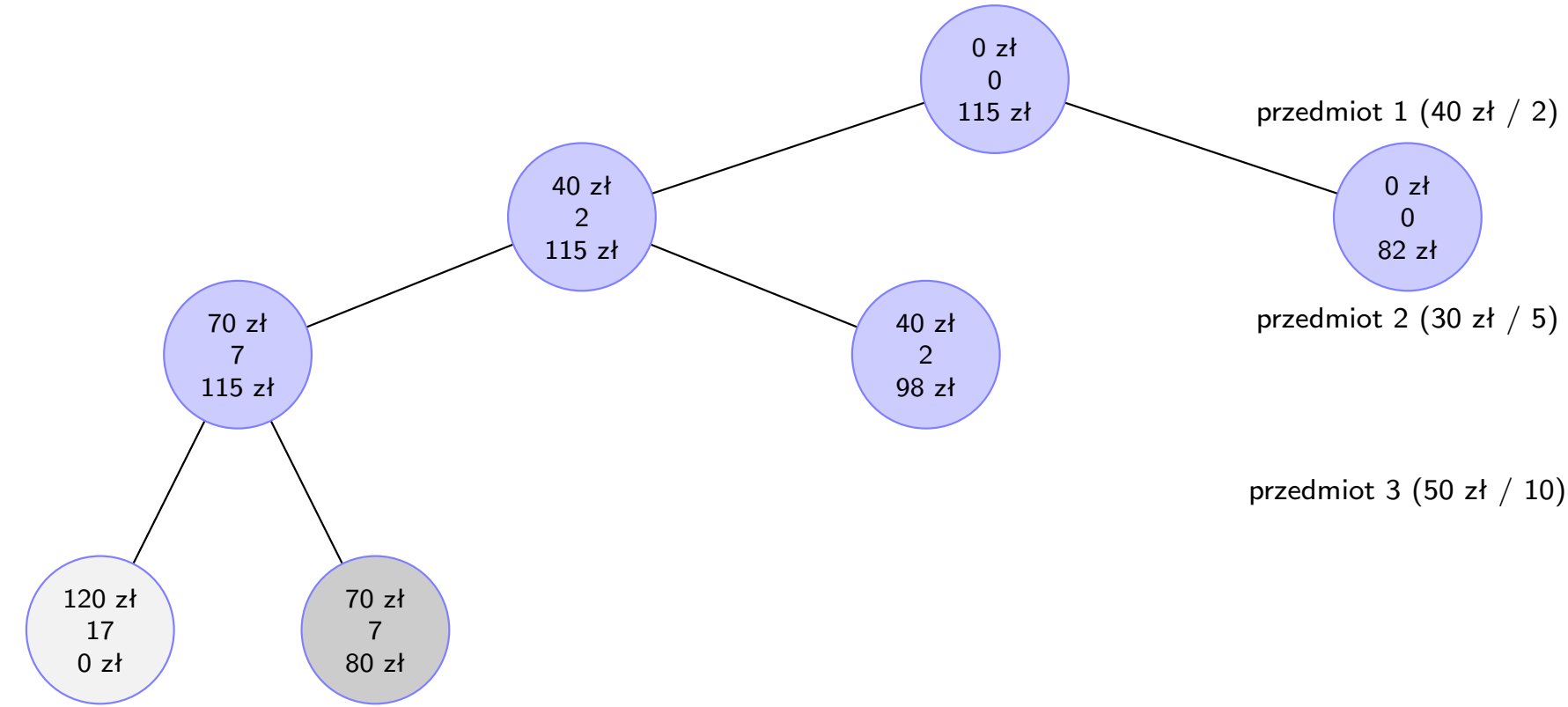
$maxprofit = 70$



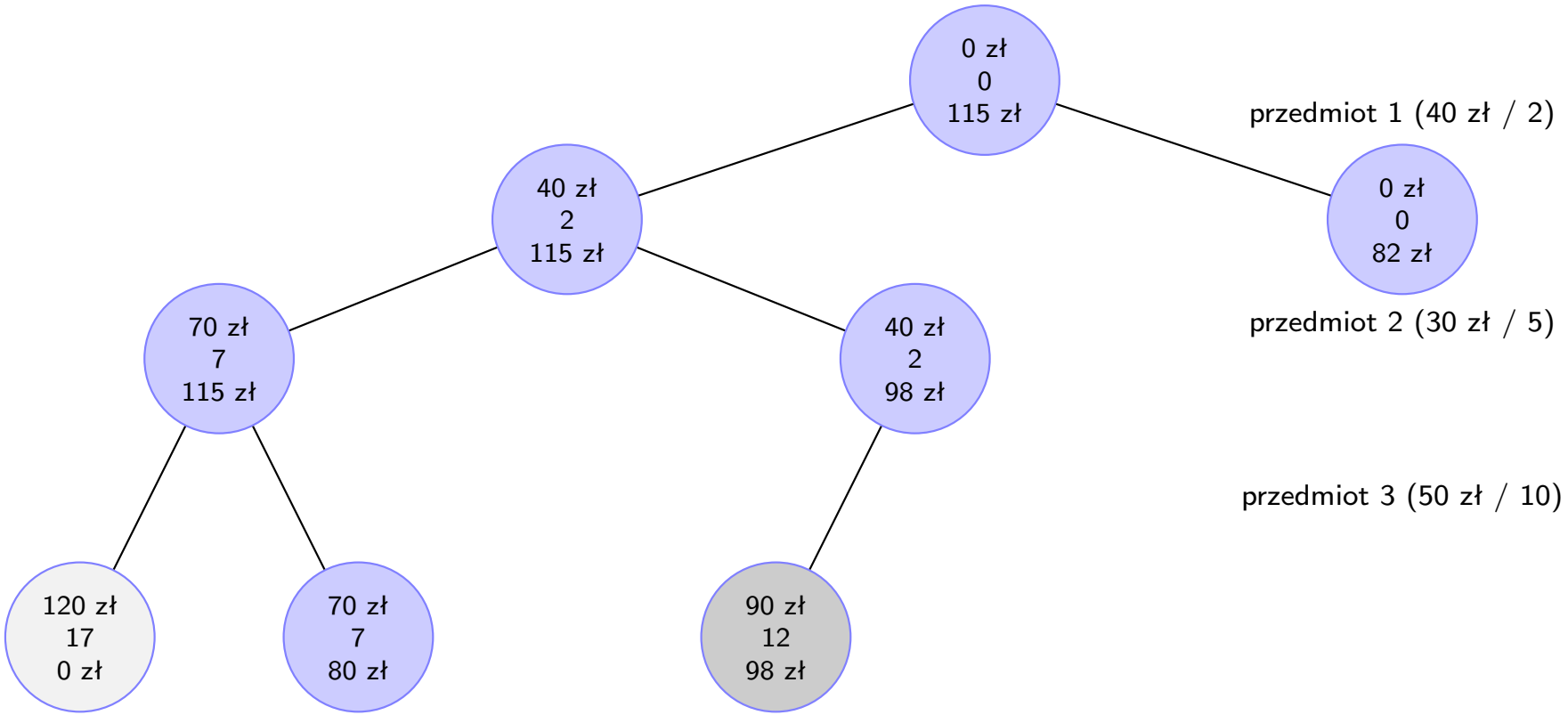
$maxprofit = 70$



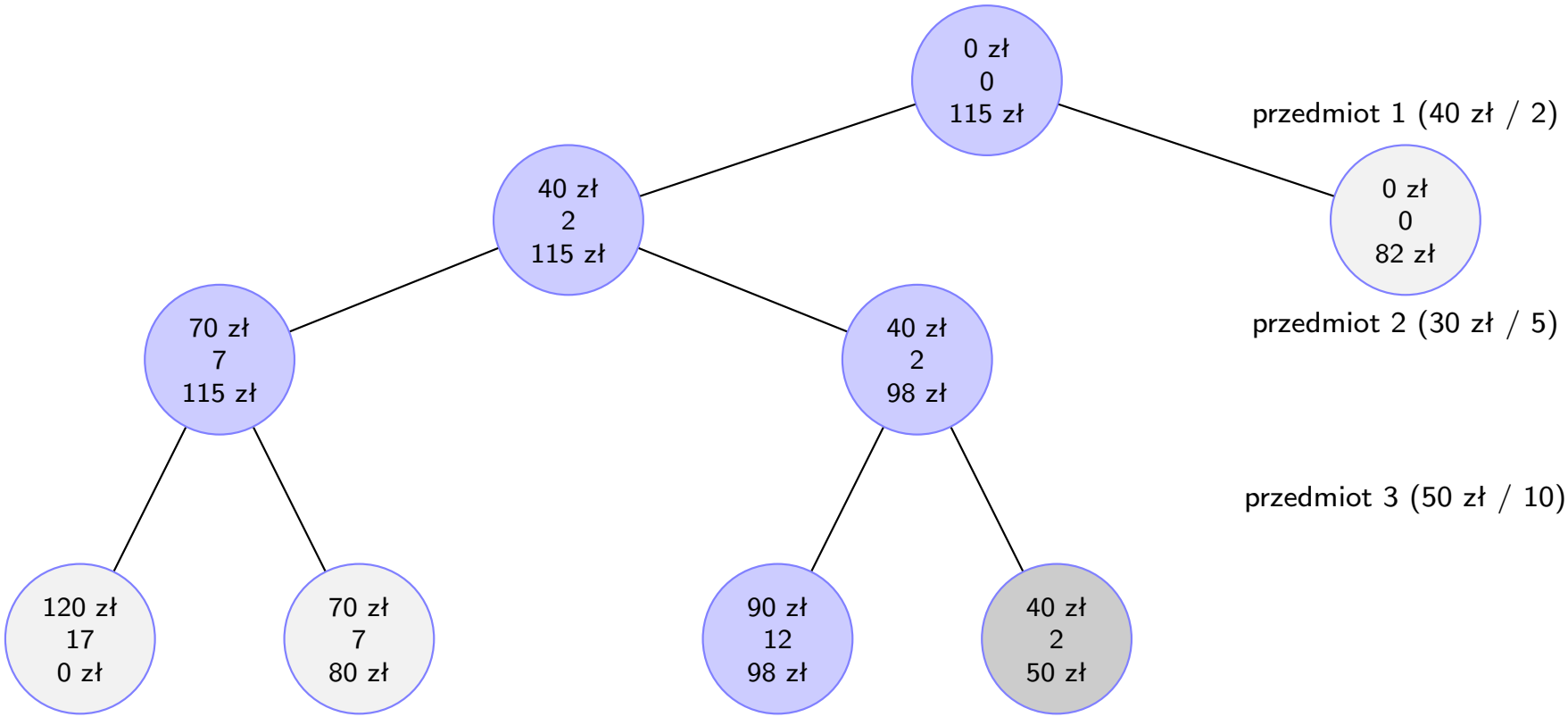
$maxprofit = 70$



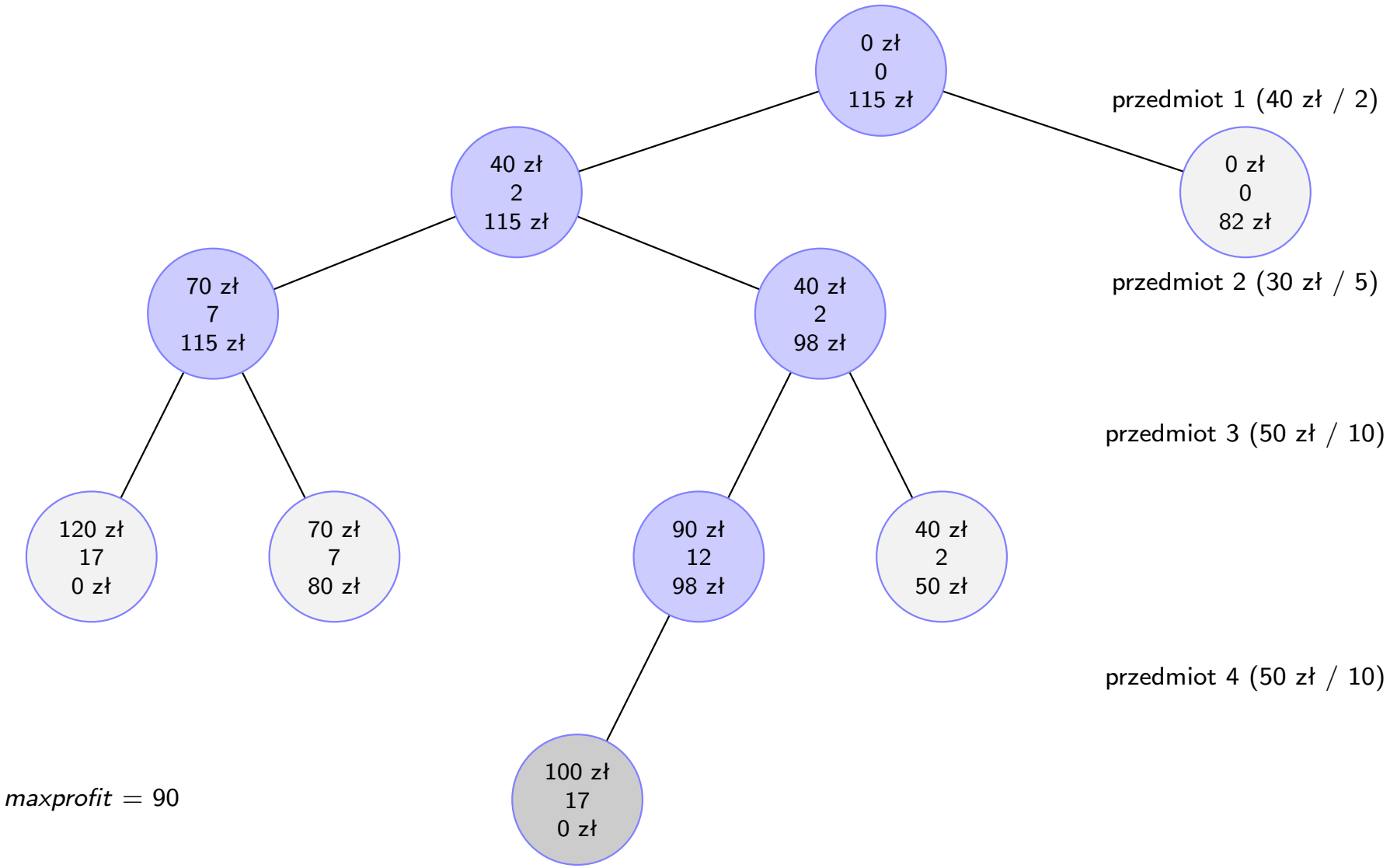
$maxprofit = 70$

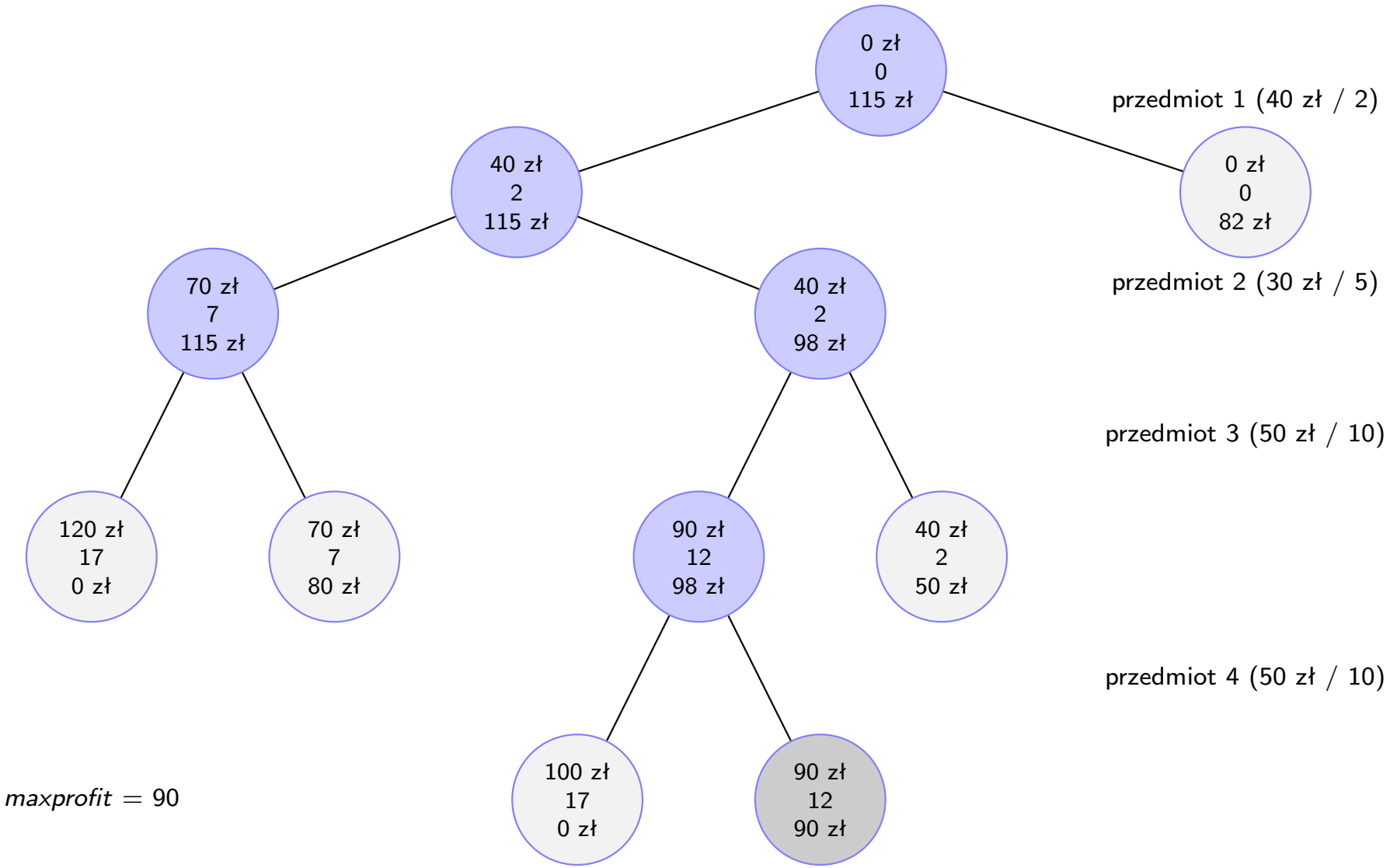


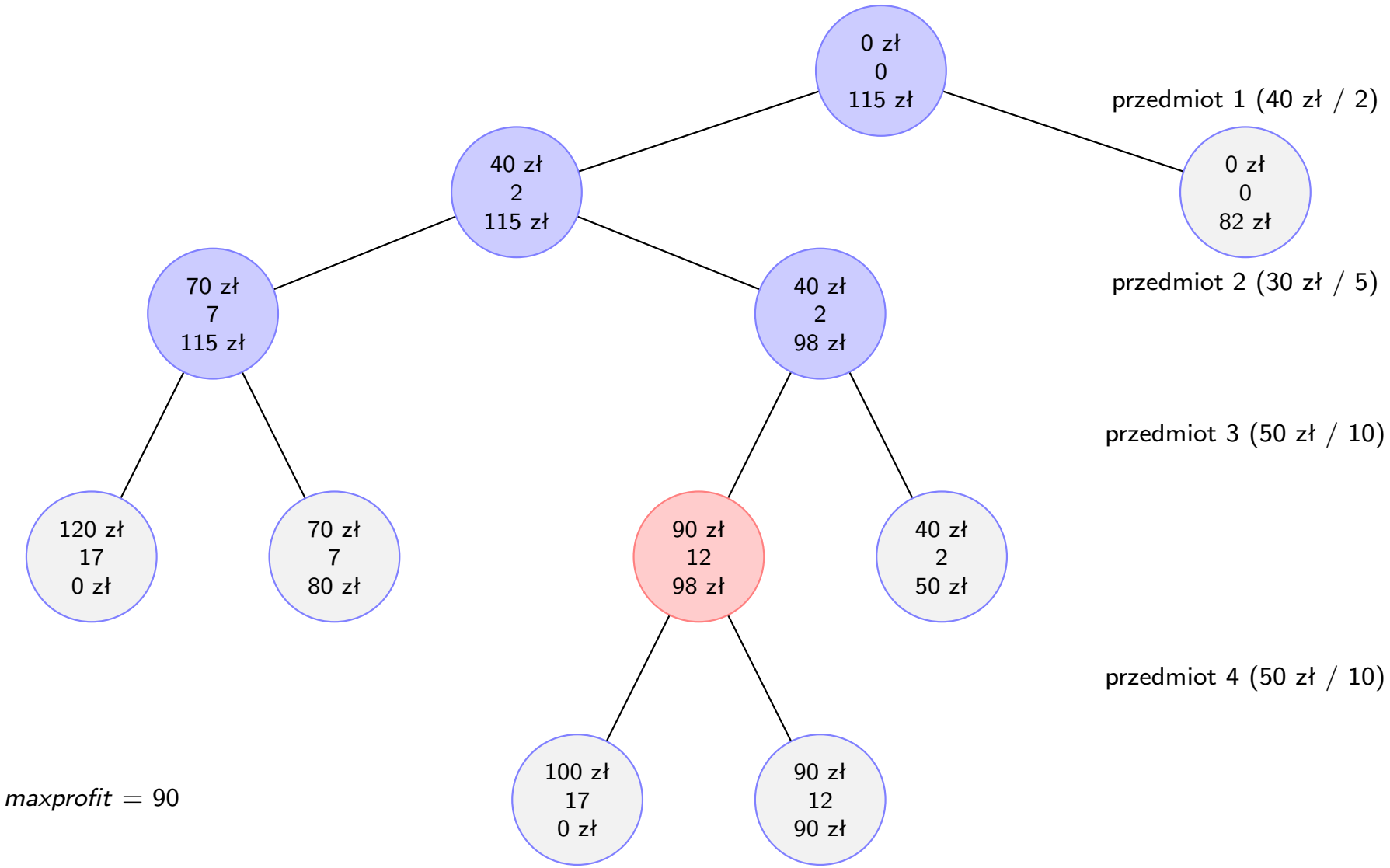
$maxprofit = 90$



$maxprofit = 90$


0 zł
0
115 zł
przedmiot 1 (40 zł / 2)
40 zł
2
115 zł
0 zł
0
82 zł
przedmiot 2 (30 zł / 5)
70 zł
7
115 zł
40 zł
2
98 zł
przedmiot 3 (50 zł / 10)
120 zł
17
0 zł
70 zł
7
80 zł
90 zł
12
98 zł
40 zł
2
50 zł
przedmiot 4 (50 zł / 10)
100 zł
17
0 zł
maxprofit = 90


0 zł
0
115 zł
przedmiot 1 (40 zł / 2)
40 zł
2
115 zł
0 zł
0
82 zł
przedmiot 2 (30 zł / 5)
70 zł
7
115 zł
40 zł
2
98 zł
przedmiot 3 (50 zł / 10)
120 zł
17
0 zł
70 zł
7
80 zł
90 zł
12
98 zł
40 zł
2
50 zł
przedmiot 4 (50 zł / 10)
100 zł
17
0 zł
90 zł
12
90 zł
maxprofit = 90


0 zł
0
115 zł
przedmiot 1 (40 zł / 2)
40 zł
2
115 zł
0 zł
0
82 zł
przedmiot 2 (30 zł / 5)
70 zł
7
115 zł
40 zł
2
98 zł
przedmiot 3 (50 zł / 10)
120 zł
17
0 zł
70 zł
7
80 zł
90 zł
12
98 zł
40 zł
2
50 zł
przedmiot 4 (50 zł / 10)
100 zł
17
0 zł
90 zł
12
90 zł
maxprofit = 90

Najpierw najlepszy

- sprawdziliśmy tylko 11 węzłów (o 6 mniej niż w przypadku przeszukiwania wszerz)
- ta strategia szybciej kieruje się do optymalnego rozwiązania
- nie ma gwarancji, że węzeł który wygląda na najlepszy da w wyniku optymalne rozwiązanie (przykład wyżej)

Algorytm

initialize(Q) (*kolejka priorytetowa*)
$v \leftarrow$ korzeń drzewa (*drzewo istnieje niejawnie*)
best $\leftarrow$ *value*(v)
insert(Q, v)
while !*empty*(Q) **do**
  $v \leftarrow$ *remove*(Q)
  if *bound*(v) jest lepszy od *best* **then**
    for all dziecko u węzła v **do**
      if *value*(u) jest lepsza od *best* **then**
        best $\leftarrow$ *value*(u)
      end if
      if *bound*(u) jest lepsza od *best* **then**
        insert(Q, u)
      end if
    end for
  end if
end while

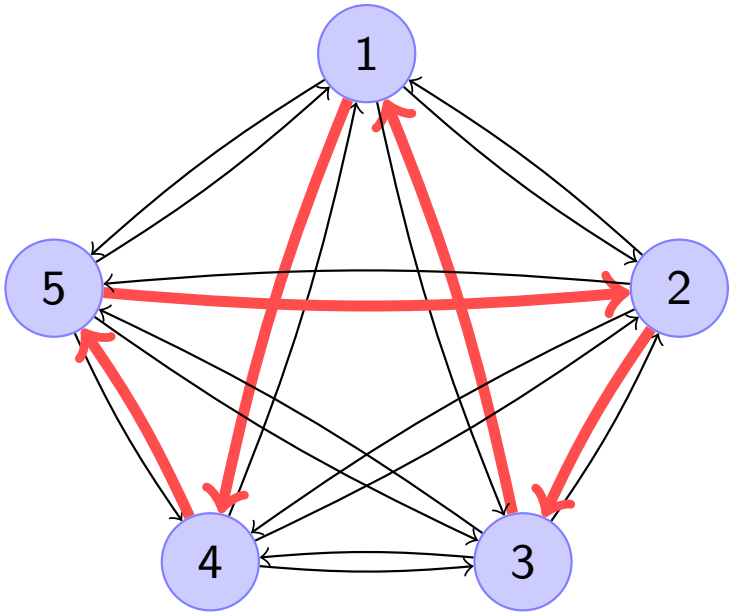
Problem komiwojażera

Definicja

- mamy graf pełny skierowany $G = (V, E; w)$, $|V| = n$
- w określa wagę krawędzi (odległość)

Problem: znaleźć cykl Hamiltona o najmniejszej wadze w G

$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



Problem komiwojażera

Naturalnym drzewem przestrzeni stanów będzie drzewo które:

- w korzeniu mamy ścieżkę długości 1 (miasto początkowe)
- na pierwszym poziomie testujemy wszystkie ścieżki o długości 2 (dokładając do miasta początkowego po kolei każde z jego sąsiadów)
- na kolejnych poziomach dodajemy kolejne miasto do ścieżki
- w liściach znajdują się ścieżki długości $n - 1$, bo ostatnie miasto jest już wtedy jednoznacznie określone

Problem komiwojażera - granica

Musimy określić dolną granicę długości ścieżek, których prefiksem jest ścieżka w obecnym węźle

Zauważmy, że:

- długość trasy nie może być krótsza niż suma minimalnych długości krawędzi wychodzących z każdego wierzchołka
- minimalną długość krawędzi wychodzących z wierzchołka i możemy wyznaczyć przeglądając i-ty wiersz macierzy sąsiedztwa

Problem komiwojażera - granica - przykład

dla korzenia

$v_1 : min(14, 4, 10, 20) = 4$
$v_2 : min(14, 7, 8, 7) = 7$
$v_3 : min(4, 5, 7, 16) = 4$
$v_4 : min(11, 7, 9, 2) = 2$
$v_5 : min(18, 7, 17, 4) = 4$

$granica = 4 + 7 + 4 + 2 + 4 = 21$

dla wierzchołka z trasą $[1, 2]$

$v_1 : 14$
$v_2 : min(7, 8, 7) = 7$
$v_3 : min(4, 7, 16) = 4$
$v_4 : min(11, 9, 2) = 2$
$v_5 : min(18, 17, 4) = 4$

$granica = 14 + 7 + 4 + 2 + 4 = 31$

Problem komiwojażera - algorytm

- zmienną *minlength* inicjujemy jako ∞
- aktualizujemy ją tylko jak dojdziemy do liścia (bo tylko tam możemy obliczyć długość całej trasy)
- nie obliczamy więc granic dla liści tylko długość trasy

Obliczanie granicy

- funkcja obliczająca granicę jest "sercem" *metody podziału i ograniczeń*
- od jej jakości zależy szybkość działania algorytmu
- obliczane granice powinny być jak najbliższe optymalnej wartości dla pod problemów
- często istnieje wiele sposobów na obliczenie granicy

Problem komiwojażera - inna granica

Możemy zauważyć, że do każdego wierzchołka wchodzi i wychodzi się dokładnie raz, dlatego:

- dla każdej krawędzi połowę jej wagi skojarzamy z wierzchołkiem z którego wychodzi, a drugą połowę z wierzchołkiem do którego wchodzi
- dlatego dla każdego wierzchołka obliczamy minimalny koszt jego odwiedzenia jako sumę połowy minimalnych kosztów wejścia i wyjścia z niego
- minimalny koszt wejścia do i-tego wierzchołka obliczamy wyznaczając minimum z i-tej kolumny
- minimalny koszt wyjścia z i-tego wierzchołka obliczamy wyznaczając minimum z i-tego wiersza

Problem komiwojażera - inna granica - przykład

Wyznaczamy początkową granicę trasy.

Obliczamy minimalny koszt odwiedzenia wierzchołka v_2:

$$\frac{min(14,5,7,7) + min(14,7,8,7)}{2} = 6$$

Kilka funkcji obliczających granice

- czasami (tak jak w przypadku problemu komiwojażera) możemy wyznaczyć kilka funkcji obliczających granice
- możemy obliczać wszystkie granice dla każdego węzła i brać pod uwagę najlepszą z nich
- nie zawsze daje to efekty ze względu na koszty obliczania granic

Bibliografia

Neapolitan R., Naimipour K.:

Podstawy algorytmów z przykładami w C++ (rozdział 6.)

2004

Jens Clausen:

Branch and Bound Algorithms - Principles and Examples

1999

Dziękuję za uwagę