



УНИВЕРСИТЕТ ИТМО

Лекция 7

Жадные алгоритмы



Жадный алгоритм (англ. *Greedy algorithm*) — алгоритм, заключающийся в принятии наилучшего на каждом этапе решения, допуская, что конечное решение также окажется оптимальным.

«Сдача»

У нас есть монеты 50, 10, 5, 1 копейка. Надо вернуть сдачу 77 копеек.

Мы берем одну монету 50 копеек, две монеты 10 копеек, одну монету 5 копеек и две монеты 1 копейку.

Нам удалось быстро определить перечень нужных монет и составить самый короткий список из монет, чтобы набрать требуемую сумму.

Основные свойства жадных алгоритмов

1. Свойство оптимальной подструктуры

Задача обладает оптимальной подструктурой, если оптимальное решение задачи может быть построено из оптимальных решений её подзадач.

Пример: В задаче о выборе активностей (Activity Selection Problem) оптимальное расписание можно построить, выбирая каждую следующую активность, которая не пересекается с уже выбранными.

```
def activity_selection(activities):
    # Сортируем активности по времени окончания
    activities.sort(key=lambda x: x[1])

    # Выбираем первую активность
    selected = [activities[0]]

    # Проходим по оставшимся активностям
    for i in range(1, len(activities)):
        if activities[i][0] >= selected[-1][1]:
            selected.append(activities[i])

    return selected

# Пример входных данных: список активностей в формате (start, end)
activities = [
    (1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8, 12), (2, 13), (12, 14)
]

# Вызов функции
selected_activities = activity_selection(activities)

# Вывод результата
print("Выбранные активности:")
for activity in selected_activities:
    print(f"Начало: {activity[0]}, Окончание: {activity[1]}")
```



2. Свойство жадного выбора

Жадный выбор — это выбор локально оптимального решения на каждом шаге с надеждой, что это приведет к глобально оптимальному решению.

Пример: В задаче о минимальном количестве монет для размена жадный алгоритм выбирает наибольшую возможную монету на каждом шаге.

```
def min_coins_greedy(amount, coins):
    # Сортируем монеты по убыванию
    coins.sort(reverse=True)

    # Список для хранения выбранных монет
    selected_coins = []

    # Проходим по всем монетам
    for coin in coins:
        # Пока текущая монета может быть использована
        while amount >= coin:
            # Уменьшаем сумму на номинал монеты
            amount -= coin
            # Добавляем монету в список выбранных
            selected_coins.append(coin)

    # Если сумма не равна нулю, значит размен невозможен
    if amount != 0:
        return "Размен невозможен"

    return selected_coins

# Пример входных данных
coins = [1, 2, 5, 10, 20, 50, 100] # Номиналы монет
amount = 123 # Сумма для размена

# Вызов функции
result = min_coins_greedy(amount, coins)

# Вывод результата
if isinstance(result, list):
    print(f"Минимальное количество монет: {len(result)}")
    print(f"Использованные монеты: {result}")
else:
    print(result)
```



3. Свойство жадной замены.

Любое частичное решение можно модифицировать, заменяя выбранные элементы на более выгодные.

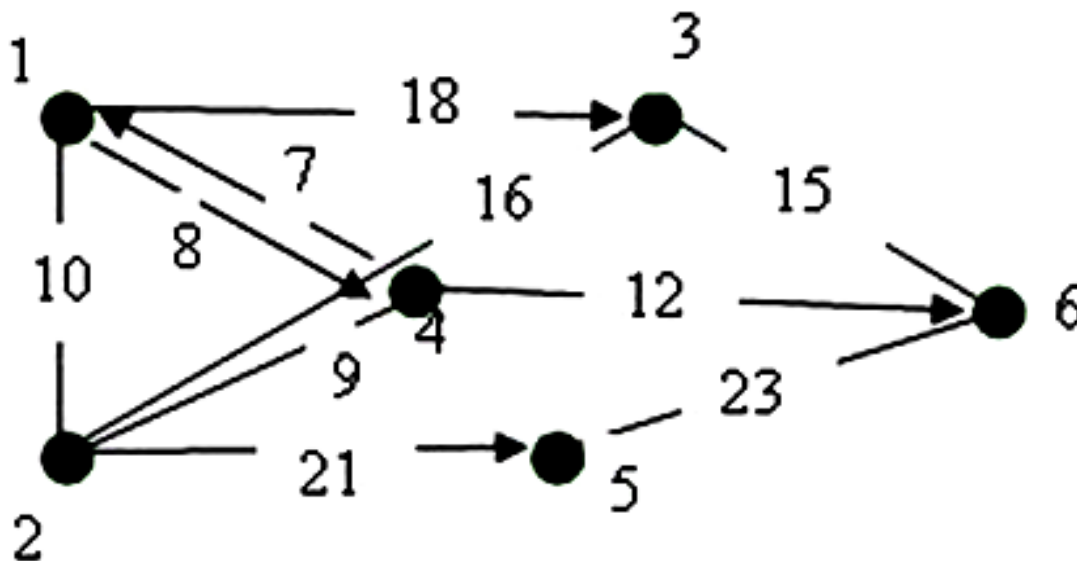
Пример: Дано множество интервалов, каждый из которых представляет собой отрезок на прямой (например, временные интервалы). Необходимо выбрать минимальное количество точек (станций), чтобы каждый интервал содержал хотя бы одну точку.

```
def min_stations(intervals):  
    # Сортируем интервалы по времени окончания  
    intervals.sort(key=lambda x: x[1])  
  
    # Список для хранения выбранных точек  
    stations = []  
  
    # Проходим по всем интервалам  
    for interval in intervals:  
        # Если текущий интервал не покрыт последней выбранной точкой  
        if not stations or interval[0] > stations[-1]:  
            # Добавляем точку в конец текущего интервала  
            stations.append(interval[1])  
  
    return stations  
  
# Пример входных данных: список интервалов в формате (start, end)  
intervals = [  
    (1, 4), (2, 5), (3, 6), (5, 7), (6, 8), (8, 10)  
]  
  
# Вызов функции  
selected_stations = min_stations(intervals)  
  
# Вывод результата  
print("Минимальное количество станций:", len(selected_stations))  
print("Расположение станций:", selected_stations)
```

Алгоритм Дейкстры — алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных.

Алгоритм Дейкстры также является одним из примеров реализации жадного алгоритма.

Необходимо найти все кратчайшие пути от вершины №1 для графа, представленного на рисунке:





Матрица кратчайших дуг:

$$\begin{bmatrix} - & 10 & 18 & 8 & - & - \\ 10 & - & 16 & 9 & 21 & - \\ - & 16 & - & - & 15 & - \\ 7 & 9 & - & - & - & 12 \\ - & - & - & - & - & 23 \\ - & - & 15 & - & 23 & - \end{bmatrix}, \text{ где «-» - } INF$$

Строка/Столбец – вершины, значения матрицы – вес ребра.

Задаем стартовые условия:

$$d(1)=0, d(x)=-$$

Окрашиваем вершину 1, $y=1$.



Находим ближайшую вершину к окрашенной нами, используя формулу:

$$d(x) = \min\{d(x); d(y) + a_{y,x}\}$$

$$d(2) = \min\{d(2); d(1) + a(1,2)\} = \min\{-; 0 + 10\} = 10$$

$$d(3) = \min\{d(3); d(1) + a(1,3)\} = \min\{-; 0 + 18\} = 18$$

$$d(4) = \min\{d(4); d(1) + a(1,4)\} = \min\{-; 0 + 8\} = 8$$

$$d(5) = \min\{d(5); d(1) + a(1,5)\} = \min\{-; 0 + -\} = -$$

$$d(6) = \min\{d(6); d(1) + a(1,6)\} = \min\{-; 0 + \infty\} = -$$

Минимальную длину имеет путь от вершины 1 до вершины 4 $d(4)=8$. Включаем вершину №4 в текущее ориентированное дерево, а так же дугу ведущую в эту вершину. Согласно выражению это дуга (1,4)

$$d(2) = \min\{d(2); d(4) + a(4,2)\} = \min\{10; 8 + 9\} = 10$$

$$d(3) = \min\{d(3); d(4) + a(4,3)\} = \min\{18; 8 + -\} = 18$$

$$d(5) = \min\{d(5); d(4) + a(4,5)\} = \min\{-; 8 + \infty\} = -$$

$$d(6) = \min\{d(6); d(4) + a(4,6)\} = \min\{-; 8 + 12\} = 20$$



Минимальную длину имеет путь от вершины 1 до вершины 2 $d(2)=10$. Включаем вершину №2 в текущее ориентированное дерево, а так же дугу ведущую в эту вершину. Согласно выражению это дуга (1,2)

$$d(3)=\min\{d(3);d(2)+a(2,3)\}=\min\{18;10+16\}=18$$

$$d(5)=\min\{d(5);d(2)+a(2,5)\}=\min\{-;10+21\}=31$$

$$d(6)=\min\{d(6);d(2)+a(2,6)\}=\min\{20;10+-\}=20$$

Минимальную длину имеет путь от вершины 1 до вершины 3 $d(3)=18$. Включаем вершину №3 в текущее ориентированное дерево, а так же дугу ведущую в эту вершину. Согласно выражению это дуга (1,3)

$$d(5)=\min\{d(5);d(3)+a(3,5)\}=\min\{31;18+15\}=31$$

$$d(6)=\min\{d(6);d(3)+a(3,6)\}=\min\{20;18+-\}=20$$



Минимальную длину имеет путь от вершины 1 до вершины 6 $d(6)=20$. Включаем вершину №6 в текущее ориентированное дерево, а так же дугу ведущую в эту вершину. Согласно выражению это дуга (4,6)

$$d(5)=\min\{d(5) ; d(6)+a(6,5)\}=\min\{31; 20+23\}=31$$

Минимальную длину имеет путь от вершины 1 до вершины 5 $d(5)=31$. Включаем вершину №5 в текущее ориентированное дерево, а так же дугу ведущую в эту вершину. Согласно выражению это дуга (2,5)

