

Динамическое программирование для «Рюкзака»

Н. Н. Кузюрин С. А. Фомин

3 декабря 2010 г.



Задача

«Сумма размеров» («Рюкзак-выполнимость»)

Даны:

$a_1, \dots, a_n, a_j \in \mathbb{N}$ — «размеры» или «веса»;

$B \in \mathbb{N}$ — «размер рюкзака».

Существует ли решение уравнения:

$$\sum_{i=1}^n a_i x_i = B, \quad x_i \in \{0, 1\}.$$

Моё хобби:

встраивать NP-полные задачи в ресторанные заказы.

Ресторан Шоткис	
Закуски	
Фруктовый салат	86
Картошка фри	110
Овощная нарезка	134
Жареные крылья	142
Сырные палочки	168
Мясное ассорти	232
Сендвичи	
Барбекю	262

Мы хотели бы что-нибудь из закусок ровно на 606 рублей.

...ровно? Эм...

Вот некоторые статьи по задаче о ранце, которые могут помочь.

Вы знаете, у меня ещё шесть других столов и
и как можно быстрее, конечно.
Хотите что-нибудь по задаче коммивояжёра?



Суммы размеров: динамическое программирование

```
def KnapsackSat(A,B):
    T = [0]
    for a in A:
        news = []
        for x in T:
            new = x + a
            if (new <= B and
                new not in T):
                news.append(new)
        T = T + news

    if B in T:
        print "Solution exists"
    else:
        print "No solution"
```

```
A = [1, 3, 4, 7, 6] B = 10
+ 1 --> [0, 1]
+ 3 --> [0, 1, 3, 4]
+ 4 --> [0, 1, 3, 4, 5, 7, 8]
+ 7 --> [0, 1, 3, 4, 5, 7, 8, 10]
+ 6 --> [0, 1, 3, 4, 5, 7, 8, 10, 6, 9]
Solution exists
```

```
A = [2, 4, 6, 8] B = 9
+ 2 --> [0, 2]
+ 4 --> [0, 2, 4, 6]
+ 6 --> [0, 2, 4, 6, 8]
+ 8 --> [0, 2, 4, 6, 8]
No solution
```

Сложность — $O(nB)$.

Полиномиальность vs. Псевдополиномиальность

Рассмотрим модификацию задачи «Сумма размеров», разрешим даже отрицательные размеры.

Полиномиальность vs. Псевдополиномиальность

Рассмотрим модификацию задачи «Сумма размеров», разрешим даже отрицательные размеры.




Полиномиальность vs. Псевдополиномиальность

Рассмотрим модификацию задачи «Сумма размеров», разрешим даже отрицательные размеры.



Формально: Даны целые числа $B, a_i, \forall i \in [1 \dots n] - n^2 \leq a_i \leq n^2$.

Рассмотрим задачу «Существует ли решение в 0-1 переменных (x_1, \dots, x_n) уравнения $\sum_{i=1}^n a_i x_i = B$?»

 Существует ли полиномиальный алгоритм для этой задачи?

Задача

«0–1 Рюкзак (Knapsack)»

Даны:

$c_1, \dots, c_n, c_j \in \mathbb{N}$ — «стоимости» предметов;

$a_1, \dots, a_n, a_j \in \mathbb{N}$ — «размеры» или «веса»;

$B \in \mathbb{N}$ — «размер рюкзака».

Найти максимальное значение f^* целевой функции

$$f \equiv \sum_{i=1}^n c_i x_i \rightarrow \max$$

с ограничением на размер «рюкзака»:

$$\sum_{i=1}^n a_i x_i \leq B, \quad x_i \in \{0, 1\}.$$

«Рюкзак»: динамическое программирование

```
def knapsack_dynprog(items, B):  
    T = {0: ItemSet()}      # Вес -> самый дорогой набор  
  
    for item in items:     # Цикл по всем предметам  
        news = []  
        for sol in T.values(): # по всем частичным  
            test = sol + item # формируем новый набор  
            if test.weight <= B and (test.weight not in T  
                or test.cost > T[test.weight].cost):  
                news.append(test) # подходит!  
  
        for sol in news: # регистрируем новые решения  
            T[sol.weight] = sol  
  
    result = ItemSet()  
    for sol in T.values(): # ищем самое  
        if sol.cost > result.cost: # дорогое решение  
            result = sol  
    return result, len(T)
```

«Рюкзак»: динамическое программирование

Предметы ($\frac{\text{СТОИМОСТЬ}}{\text{ВЕС}}$): $[\frac{6}{3}, \frac{3}{4}, \frac{2}{5}, \frac{5}{6}, \frac{5}{7}, \frac{1}{8}]$, $B = 9$

Sols	item	news
$0: \frac{0}{0}$	$\frac{6}{3}$	$[\frac{6}{3}]$
$0: \frac{0}{0}, 3: \frac{6}{3}$	$\frac{3}{4}$	$[\frac{3}{4}, \frac{9}{7}]$
$0: \frac{0}{0}, 3: \frac{6}{3}, 4: \frac{3}{4}, 7: \frac{9}{7}$	$\frac{2}{5}$	$[\frac{2}{5}, \frac{8}{8}, \frac{5}{9}]$
$0: \frac{0}{0}, 3: \frac{6}{3}, 4: \frac{3}{4}, 5: \frac{2}{5}, 7: \frac{9}{7}, 8: \frac{8}{8}, 9: \frac{5}{9}$	$\frac{5}{6}$	$[\frac{5}{6}, \frac{11}{9}]$
$0: \frac{0}{0}, 3: \frac{6}{3}, 4: \frac{3}{4}, 5: \frac{2}{5}, 6: \frac{5}{6}, 7: \frac{9}{7}, 8: \frac{8}{8}, 9: \frac{11}{9}$	$\frac{5}{7}$	$[\]$
$0: \frac{0}{0}, 3: \frac{6}{3}, 4: \frac{3}{4}, 5: \frac{2}{5}, 6: \frac{5}{6}, 7: \frac{9}{7}, 8: \frac{8}{8}, 9: \frac{11}{9}$	$\frac{1}{8}$	$[\]$

Оптимальное решение: $\frac{11}{9}$

Полиномиальность vs. Псевдополиномиальность

Лемма

Сложность алгоритма с отбором «дорогих» решений — $O(nB)$.

Упражнение

Придумайте входные наборы для этого алгоритма, на которых он будет работать экспоненциальное время.

Упражнение

Постройте алгоритм динамического программирования для задачи «Knapsack», основанный на отборе наиболее «легких» частичных решений.

- Какова будет его временная сложность?
- Придумайте входные наборы для этого алгоритма, на которых он будет работать экспоненциальное время.

«Рюкзак»: отбор легких решений

```
def knapsack_dynprog_lightest(items, B):  
    T = {0: ItemSet()} # Цена -> самый легкий набор  
  
    for item in items: # Цикл по всем предметам  
        newT = []  
        for sol in T.values(): # по всем частичным  
            test = sol + item # формируем новый набор  
            if test.weight <= B and (test.cost not in T  
                or test.weight < T[test.cost].weight):  
                newT.append(test) # подходит!  
  
        for sol in newT: # регистрируем  
            T[sol.cost] = sol # новые решения  
  
    return T[max(T.keys())] # возвращаем самое дорогое
```

«Рюкзак»: отбор легких решений

Предметы ($\frac{\text{СТОИМОСТЬ}}{\text{ВЕС}}$): $[\frac{6}{3}, \frac{3}{4}, \frac{2}{5}, \frac{5}{6}, \frac{5}{7}, \frac{1}{8}]$, $B = 9$

T	item	newT
$0: \frac{0}{0}$	$\frac{6}{3}$	$[\frac{6}{3}]$
$0: \frac{0}{0}, 6: \frac{6}{3}$	$\frac{3}{4}$	$[\frac{3}{4}, \frac{9}{7}]$
$0: \frac{0}{0}, 9: \frac{9}{7}, 3: \frac{3}{4}, 6: \frac{6}{3}$	$\frac{2}{5}$	$[\frac{2}{5}, \frac{5}{9}, \frac{8}{8}]$
$0: \frac{0}{0}, 2: \frac{2}{5}, 3: \frac{3}{4}, 5: \frac{5}{9}, 6: \frac{6}{3}, 8: \frac{8}{8}, 9: \frac{9}{7}$	$\frac{5}{6}$	$[\frac{5}{6}, \frac{11}{9}]$
$0: \frac{0}{0}, 2: \frac{2}{5}, 3: \frac{3}{4}, 5: \frac{5}{6}, 6: \frac{6}{3}, 8: \frac{8}{8}, 9: \frac{9}{7}, 11: \frac{11}{9}$	$\frac{5}{7}$	$[\]$
$0: \frac{0}{0}, 2: \frac{2}{5}, 3: \frac{3}{4}, 5: \frac{5}{6}, 6: \frac{6}{3}, 8: \frac{8}{8}, 9: \frac{9}{7}, 11: \frac{11}{9}$	$\frac{1}{8}$	$[\frac{1}{8}]$

Оптимальное решение: $\frac{11}{9}$

Парето-оптимальные решения

Определение

Пусть S_1 и S_2 допустимые подмножества предметов для задачи «Knapsack». S_1 **доминирует** над S_2 , если:

- стоимость S_1 больше стоимости S_2 ,
- вес S_1 не больше веса S_2 .

т. е. набор доминирующих подмножеств есть набор *Парето-оптимальных* решений, т. е. таких решений, в которых нельзя улучшить один параметр (стоимость), без ухудшения другого параметра (увеличения веса).

Алгоритм Немхаузера – Ульмана

```
def KnapsackNemhauserUllman(items, B):  
    pareto = [ItemSet()] # Парето-оптимальные по весу  
    for item in items:  
        news = []  
        for solution in pareto:  
            if solution.weight + item.weight <= B:  
                news.append(solution + item)  
        mergedItemSet = merge_item_sets(pareto, news)  
        pareto = mergedItemSet  
    return pareto[-1], len(pareto)
```

Слияние доминирующих решений

```
def merge_item_sets(s1, s2): # Слияние списков доминирующих решений
    i = j = 0 # можем считать, что входные списки непусты.
    res = []
    while i <= len(s1) or j <= len(s2): # пока не дошли до конца
        if i == len(s1):
            res += s2[j:] # забираем остаток второго списка
            break
        if j == len(s2):
            res += s1[i:] # забираем остаток первого списка
            break

        if s1[i].weight <= s2[j].weight:
            if s1[i].cost > s2[j].cost:
                j += 1
            else:
                res.append(s1[i])
                i += 1
        else:
            if s1[i].cost < s2[j].cost:
                i += 1
            else:
                res.append(s2[j])
                j += 1

    return res
```


Алгоритм Немхаузера – Ульмана

Предметы ($\frac{\text{СТОИМОСТЬ}}{\text{ВЕС}}$): $[\frac{6}{3}, \frac{3}{4}, \frac{2}{5}, \frac{3}{3}, \frac{6}{8}]$, $B = 10$

pareto	news	mergedItemSet
$[\frac{0}{0}]$	$[\frac{6}{3}]$	$[\frac{0}{0}, \frac{6}{3}]$
$[\frac{0}{0}, \frac{6}{3}]$	$[\frac{3}{4}, \frac{9}{7}]$	$[\frac{0}{0}, \frac{6}{3}, \frac{9}{7}]$
$[\frac{0}{0}, \frac{6}{3}, \frac{9}{7}]$	$[\frac{2}{5}, \frac{8}{8}]$	$[\frac{0}{0}, \frac{6}{3}, \frac{9}{7}]$
$[\frac{0}{0}, \frac{6}{3}, \frac{9}{7}]$	$[\frac{3}{3}, \frac{9}{6}, \frac{12}{10}]$	$[\frac{0}{0}, \frac{6}{3}, \frac{9}{6}, \frac{9}{7}, \frac{12}{10}]$
$[\frac{0}{0}, \frac{6}{3}, \frac{9}{6}, \frac{9}{7}, \frac{12}{10}]$	$[\frac{6}{8}]$	$[\frac{0}{0}, \frac{6}{3}, \frac{9}{6}, \frac{9}{7}, \frac{12}{10}]$

Оптимальное решение: $\frac{12}{10}$

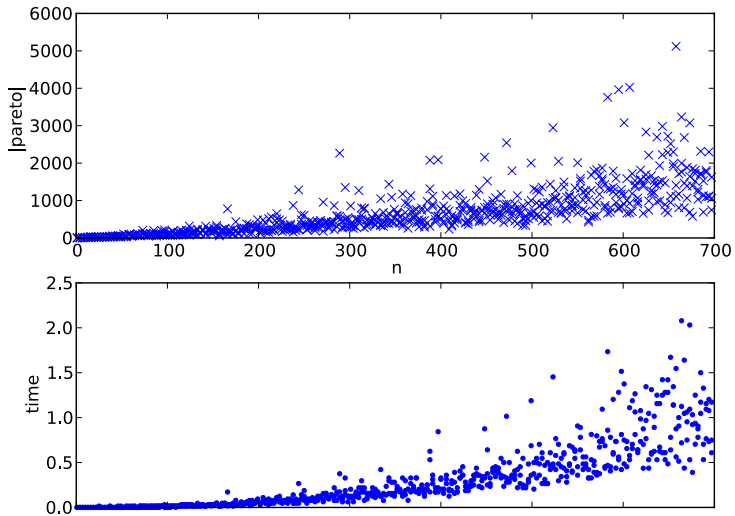
Алгоритм Немхаузера – Ульмана

Упражнение

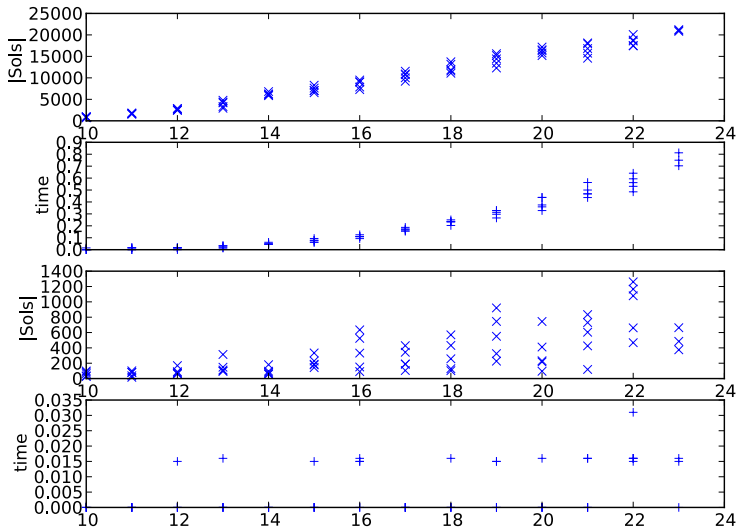
Придумайте входные наборы для алгоритма Немхаузера – Ульмана, на которых он будет работать экспоненциальное время.

Заметим, что практика использования показала, что на реальных данных алгоритм Немхаузера – Ульмана работает достаточно хорошо (см. тему «Полиномиальный в среднем алгоритм для задачи о рюкзаке»).

Алгоритм «Немхаузера-Ульмана» на случайных данных



Алгоритм «Рюкзак ДинПрог» на случайных данных



<http://discopal.ispras.ru/>

Вопросы?