

Tên: Châu Tấn

MSSV: 20520926

Lớp: CS106.M21 (Môn: Trí tuệ nhân tạo)

GVHD: TS. Lương Ngọc Hoàng



BÁO CÁO BÀI TẬP 2:

SOLVING KNAPSACK PROBLEM VỚI GOOGLE OR TOOLS

Yêu cầu:

1) Solving knapsack with OR-Tools: <https://developers.google.com/optimization/bin/knapsack>

2) Knapsack test instances: <https://github.com/likr/kplib>

Có tổng cộng 13 nhóm test cases (00-12) cho bài toán knapsack trong link [2]. Trong mỗi nhóm, chúng ta cần chọn ra ít nhất 5 test cases (càng nhiều test cases càng tốt) có kích thước khác nhau (ví dụ 50 items, 100 items, 200 items, 500 items, 1000 items,...), và giải các test cases này bằng OR Tools như sau:

1. Chọn một mốc chi phí tính toán phù hợp với máy tính của mỗi bạn (ví dụ tối đa 3 phút cho mỗi lần chạy).

2. Thiết lập thực nghiệm sao cho OR Tools sẽ dừng khi mà thời gian tính toán cho mỗi lần chạy đã sử dụng hết. (Tham khảo set_time_limit tại: https://developers.google.com/optimization/reference/algorithms/knapsack_solver/KnapsackSolver)

3. Lưu lại kết quả của mỗi lần chạy mỗi test case. Lời giải tìm ra có phải là lời giải tối ưu của test case đó hay không?

4. Lập bảng thống kê: Tên của mỗi test case, giá trị của lời giải, tổng trọng lượng các items trong lời giải, lời giải tìm ra có phải là tối ưu hay không?

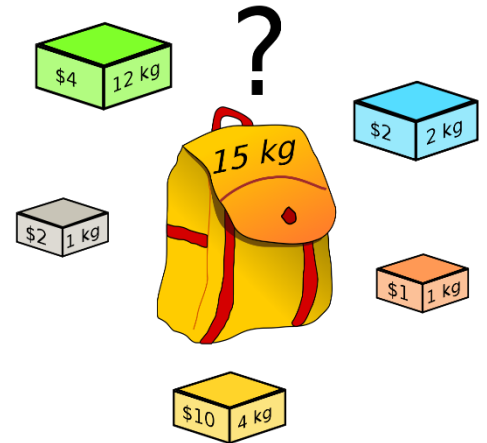
5. Dựa vào kết quả thống kê, kết luận trong 13 nhóm test cases, nhóm nào là dễ và nhóm nào là khó?

Bài làm:

1. TỔNG QUAN

1.1. KNAPSACK PROBLEMS LÀ GÌ ?

- Trong Knapsack problems, người chơi sẽ được giao cho 1 bộ những vật thể với giá trị và thể tích được biết trước. Nếu mà tổng lượng thể tích của vật thể lớn hơn thể tích của balo thì balo sẽ không thể nào chứa hết các vật thể đó. Do đó, người chơi sẽ phải bỏ lại một số vật thể lại. Bài toán ba lô là bài toán yêu cầu người chơi phải bỏ những vật thể vào balo sao cho tối ưu nhất được sức chứa của cái ba lô.
- Trong bài tập này thì em dùng thuật toán BRANCH AND BOUND được tích hợp trong OR tools để giải bài toán này.



Hình 1: Knapsack problems
(Nguồn: wikipedia.org)

1.2. THUẬT TOÁN BRANCH AND BOUND LÀ GÌ ?

- Trong lập trình cũng như trong thực tế, chắc hẳn chúng ta đều đã gặp rất nhiều những bài toán với yêu cầu tìm kết quả tốt nhất thỏa mãn một hoặc một số điều kiện nào đó. Những bài toán này được gọi chung là bài toán tối ưu.
- Thuật toán nhánh và cận (Branch and Bound) là thuật toán cải tiến từ thuật toán quay lui (Backtracking) được sử dụng để tìm nghiệm của bài toán tối ưu.

2. THỰC HIỆN GIẢI BÀI TOÁN VỚI OR-TOOLS

2.1. XỬ LÝ DỮ LIỆU CÁC TEST CASES

- Dữ liệu được cho trong đề bài là một bộ dữ liệu tốt để chạy thuật toán. Tuy nhiên bộ dữ liệu có quá nhiều test case cho nên em không thể chạy tiếp được.
- Do đó em chọn chiến lược là lấy 5 bộ $n = 50$, $n = 100$, $n = 200$, $n = 500$, $n = 1000$. Mỗi bộ thì em lấy 3 file để kiểm thử đó là s000.kp, s001.kp, s002.kp trong thư mục R01000.

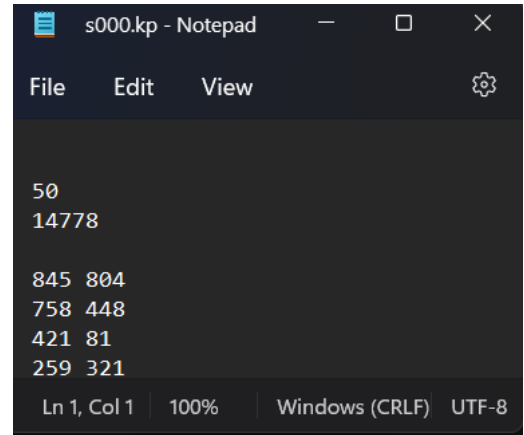
```
def store_direction(case):  
    pathfiles = []  
    root_path = 'kplib'  
    instances = os.listdir(root_path)  
    NumbersOfValues = ['n00050', 'n00100', 'n00200', 'n00500', 'n01000']  
    for files in instances:  
        for NumOfVal in NumbersOfValues:  
            pathfiles.append(f"{root_path}\\{files}\\{NumOfVal}\\R01000\\{case}")  
    return pathfiles
```

(Đoạn code của hàm store_direction(case))

- Ở hàm store_direction(case) có mục đích lưu lại các địa chỉ dẫn đến các file kiểm thử. Với case là một trong ba phần tử ['s000.kp', 's001.kp', 's002.kp'].

- Hàm `get_info(path)` sẽ nhận những địa chỉ đến file kiểm thử từ hàm `store_direction(case)`. Hàm này có mục đích trích xuất những thông tin từ file kiểm thử và trả về 4 giá trị `problem_size`(số lượng items), `capacity`(sức chứa), `values`(một mảng chứa các values), `weights`(một mảng chứa các weights)

```
def get_info(path):
    with open(path, 'r') as fi:
        content = fi.read()
        content = content.split('\n')
        problem_size = int(content[1])
        capacity = [int(content[2])]
        values = []
        weights = [[]]
        for item in content[4:-1]:
            value, weight = item.split(" ")
            values.append(int(value))
            weights[0].append(int(weight))
        return problem_size, capacity, values, weights
```



Hình 2: Bên trong file s000.kp

(Đoạn code của hàm `get_info`)

- Cấu trúc của 1 file kiểm thử. Dòng số 1 và dòng số 4 là khoảng trắng dòng số 2 là `problem_size`, dòng số 4 là `capacity`, từ dòng thứ 5 trở đi là các giá trị `value` và `weight`.

2.2. CHỌN MỨC CHI PHÍ PHÙ HỢP VÀ CÀI ĐẶT THUẬT TOÁN

- Trong bài này, em chọn `TIME_LIMIT` là 60 giây để cài đặt cho thuật toán. Để giải quyết bài toán này thì em sử dụng thuật toán Branch and bound (nhánh và cận) được tích hợp trong công cụ Google OR tools để giải quyết bài toán này.
- Để xác định được thuật toán đó có tối ưu hay không thì chúng ta dựa vào thời gian chạy của thuật toán trên test case đó. Nếu thời gian chạy trên test case đó lớn hơn `TIME_LIMIT` thì thuật toán đó không tối ưu. Trường hợp ngược lại thì nó tối ưu.
- Sau khi chạy thuật toán xong thì các giá trị sẽ được ghi nhận vào các biến `list_optimal`, `list_values` và `list_weight` dưới dạng list.

```
Cases = ['s000.kp', 's001.kp', 's002.kp']
# Create the solver.
Solver = pywrapknapsack_solver.KnapsackSolver(
    pywrapknapsack_solver.KnapsackSolver.
    KNAPSACK_MULTIDIMENSION_BRANCH_AND_BOUND_SOLVER, 'KnapsackExample')
TIME_LIMIT = 60
for case in cases:
    ListOfPathFiles = store_direction(case)
    list_weight = []
    list_values = []
    list_optimal = []
    number_values = [50, 100, 200, 500, 1000]
    for file in ListOfPathFiles:
        n, capacities, values, weights = get_info(file)
        solver.Init(values, weights, capacities)
        solver.set_time_limit(TIME_LIMIT)
        t_0 = time.time()
```

```

computed_value = solver.Solve()
t_1 = time.time() - t_0
packed_items = []
packed_weights = []
total_weight = 0
print('Total value =', computed_value)
list_values.append(computed_value)
for i in range(len(values)):
    if solver.BestSolutionContains(i):
        packed_items.append(i)
        packed_weights.append(weights[0][i])
        total_weight += weights[0][i]
print(f'Solution for {file}: ')
print('Total weight:', total_weight)
list_weight.append(total_weight)
print('Packed items:', packed_items)
print('Packed_weights:', packed_weights)
if t_1 > TIME_LIMIT:
    list_optimal.append('_')
else:
    list_optimal.append('Optimal')
print(f'Optimal or not:{list_optimal[-1]}')

```

(Đoạn code triển khai thuật toán bằng ortools)

```

PS D:\UIT\nam2\HK2\Artificial_Intelligent\assignment\knapsack_problems_Google_OR_tools> python Google_OR_tools.py
Total value = 20995
Solution for kplib\00Uncorrelated\n00050\R01000\s000.kp:
Total weight: 14721
Packed items: [0, 1, 2, 4, 6, 9, 10, 11, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 28, 29, 31, 33, 39, 42, 45, 47, 49]
Packed_weights: [804, 448, 81, 508, 110, 548, 815, 541, 604, 588, 597, 385, 576, 291, 190, 187, 613, 657, 477, 90, 924, 843, 924, 392, 590, 451, 917, 83, 487]
Optimal or not:Optimal
Total value = 46537
Solution for kplib\00Uncorrelated\n00100\R01000\s000.kp:
Total weight: 22519
Packed items: [0, 2, 4, 5, 6, 9, 10, 12, 13, 16, 17, 18, 19, 21, 22, 24, 25, 27, 28, 29, 31, 32, 33, 34, 36, 38, 39, 41, 42, 44, 45, 49, 50, 55, 57, 58, 59, 60, 61, 62, 63, 64, 65,
67, 68, 69, 72, 73, 74, 76, 77, 78, 80, 81, 83, 84, 86, 87, 88, 89, 90, 91, 93, 94, 95, 96, 99]
Packed_weights: [631, 244, 118, 221, 795, 101, 147, 46, 574, 681, 27, 635, 607, 392, 371, 37, 22, 185, 124, 211, 937, 23, 426, 102, 221, 351, 181, 40, 101, 200, 359, 170, 673, 343,
597, 443, 175, 472, 410, 570, 509, 312, 358, 251, 561, 13, 46, 281, 241, 353, 288, 360, 634, 622, 389, 415, 2, 193, 335, 240, 638, 379, 569, 415, 403, 702, 47]
Optimal or not:Optimal
Total value = 84317
Solution for kplib\00Uncorrelated\n00200\R01000\s000.kp:
Total weight: 50302
Packed items: [0, 1, 2, 4, 6, 8, 9, 10, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 28, 29, 31, 32, 33, 34, 36, 37, 38, 39, 42, 45, 47, 48, 49, 50, 55, 58, 60, 61, 62, 63, 64,
66, 67, 68, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 94, 95, 96, 98, 101, 103, 106, 107, 108, 109, 110, 111, 113, 114, 115, 116, 118, 120, 122, 1
23, 126, 130, 131, 133, 135, 136, 137, 138, 140, 143, 146, 147, 148, 150, 151, 153, 154, 155, 156, 160, 161, 162, 166, 168, 170, 171, 173, 174, 175, 176, 177, 178, 179, 181, 182, 1
83, 185, 191, 192, 193, 195, 196, 197, 198]
Packed_weights: [446, 260, 158, 488, 756, 495, 313, 467, 813, 189, 634, 84, 726, 987, 402, 679, 317, 214, 3, 529, 98, 119, 874, 280, 979, 101, 397, 82, 275, 453, 134, 348, 279, 19,
41, 681, 43, 656, 903, 641, 373, 538, 208, 9, 152, 334, 621, 42, 164, 290, 395, 549, 294, 479, 240, 49, 180, 524, 71, 404, 329, 415, 100, 909, 475, 344, 480, 700, 302, 920, 376, 6
6, 85, 750, 62, 8, 394, 449, 489, 585, 680, 369, 261, 432, 359, 703, 119, 398, 43, 216, 147, 198, 379, 152, 149, 878, 496, 918, 499, 499, 202, 610, 219, 341, 36, 149, 257, 719, 67,
869, 40, 41, 16, 844, 331, 161, 149, 657, 505, 902, 503, 679, 207, 536, 599, 483, 792, 389, 587]
Optimal or not:Optimal

```

Hình 3: Màn hình terminal sau khi chạy thuật toán

2.3. LƯU LẠI CÁC GIÁ TRỊ SAU KHI CHẠY XONG THUẬT TOÁN

```

#Data processing
values_file = []
weight_file = []
Optimal_file = []
count = 1
for index in list_values:
    if count % 5 == 0:
        features = list_values[count-5:count]
        values_file.append(features)

```










```

        count += 1
    with open(f'values({case}).csv', 'w+') as fow:
        write = csv.writer(fow)
        write.writerow(number_values)
        write.writerows(values_file)
    count = 1
    for index in list_weight:
        if count % 5 == 0:
            features = list_weight[count-5:count]
            weight_file.append(features)
        count += 1
    with open(f'weights({case}).csv', 'w+') as fow:
        write = csv.writer(fow)
        write.writerow(number_values)
        write.writerows(weight_file)
    count = 1
    for index in list_optimal:
        if count % 5 == 0:
            features = list_optimal[count-5:count]
            Optimal_file.append(features)
        count += 1
    with open(f'optimals({case}).csv', 'w+') as foo:
        write = csv.writer(foo)
        write.writerow(number_values)
        write.writerows(Optimal_file)

```

(Đoạn code để lưu các giá trị lại vào file .csv)

- Sau khi chạy xong thuật toán, thì các giá trị như weight, value và optimal cần được lưu lại. Sau một quá trình tìm hiểu thì em quyết định lưu các giá trị trên dưới dạng csv (comma seperated values).
- Ứng với mỗi case thì sẽ có 3 bảng về weights, values và optimals của các case đó.

| | | | |
|---|-------------------|-----------------------|------|
|  optimals(s000.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  optimals(s001.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  optimals(s002.kp).csv | 4/23/2022 3:52 PM | Microsoft Excel Co... | 1 KB |
|  values(s000.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  values(s001.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  values(s002.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  weights(s000.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  weights(s001.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |
|  weights(s002.kp).csv | 4/23/2022 3:51 PM | Microsoft Excel Co... | 1 KB |

Hình 4: Các file được tạo thành sau khi chạy xong thuật toán

- 3 bảng này vẫn chưa có cột instances (các màn của test case), để thêm cột instances vào thì chỉ cần chạy file add_instance.py trong file source code là được.

3. THỐNG KÊ KẾT QUẢ THỰC NGHIỆM

- Như đã nói ở phần đầu, bộ dữ liệu được đưa là một bộ dữ liệu tốt. Tuy nhiên, trong bộ dữ liệu có quá nhiều test case cho nên em không thể chạy hết được. Do đó, em chọn chiến lược là mỗi mục số lượng items thì em lấy 3 cái test case đầu tiên để chạy.

Thống kê đối với các case s000.kp

| Instances | Values | | | | | Weights | | | | | Optimals | | | | |
|-----------|-----------------|--------|---------|---------|---------|-----------------|---------|---------|----------|----------|-----------------|---------|---------|---------|---------|
| | Number of items | | | | | Number of items | | | | | Number of items | | | | |
| | 50 | 100 | 200 | 500 | 1000 | 50 | 100 | 200 | 500 | 1000 | 50 | 100 | 200 | 500 | 1000 |
| 0 | 20995 | 46537 | 84317 | 207992 | 400811 | 14721 | 22519 | 50302 | 118693 | 252480 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 1 | 15768 | 31064 | 56976 | 139258 | 273052 | 14232 | 29013 | 51563 | 127276 | 245972 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 2 | 17539 | 35617 | 65363 | 162178 | 316372 | 14239 | 29017 | 51563 | 127278 | 245972 | Optimal | Optimal | – | – | – |
| 3 | 14914 | 30468 | 54964 | 136031 | 263977 | 16714 | 33968 | 61464 | 152031 | 295477 | Optimal | Optimal | Optimal | – | – |
| 4 | 17556 | 35611 | 65385 | 162154 | 316415 | 14238 | 29016 | 51563 | 127278 | 245972 | Optimal | Optimal | – | – | Optimal |
| 5 | 14239 | 29017 | 51563 | 127278 | 245972 | 14239 | 29017 | 51563 | 127278 | 245972 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 6 | 19676 | 39791 | 75678 | 189769 | 371246 | 2401482 | 4902253 | 9904900 | 24712055 | 49525319 | Optimal | Optimal | Optimal | – | Optimal |
| 7 | 13472 | 24228 | 47836 | 114616 | 228624 | 4569 | 8748 | 17274 | 42898 | 84656 | Optimal | – | – | – | – |
| 8 | 10354 | 20550 | 40575 | 98713 | 196050 | 11452 | 20824 | 41116 | 100076 | 198664 | Optimal | Optimal | Optimal | – | Optimal |
| 9 | 28440 | 51656 | 101888 | 245144 | 488680 | 11540 | 20956 | 41288 | 99944 | 198780 | – | Optimal | – | – | – |
| 10 | 21338 | 43316 | 81658 | 203778 | 399170 | 14238 | 29016 | 51558 | 127278 | 245970 | Optimal | Optimal | Optimal | – | – |
| 11 | 14229 | 29001 | 51540 | 127239 | 245877 | 14238 | 29015 | 51562 | 127277 | 245972 | Optimal | – | – | – | – |
| 12 | 300031 | 611418 | 1086483 | 2681868 | 5182856 | 14239 | 29017 | 51563 | 127278 | 245972 | Optimal | Optimal | – | – | Optimal |

Thống kê đối với các case s001.kp

| Instances | Values | | | | | Weights | | | | | Optimals | | | | |
|-----------|-----------------|--------|---------|---------|---------|-----------------|---------|---------|----------|----------|-----------------|---------|---------|---------|---------|
| | Number of items | | | | | Number of items | | | | | Number of items | | | | |
| | 50 | 100 | 200 | 500 | 1000 | 50 | 100 | 200 | 500 | 1000 | 50 | 100 | 200 | 500 | 1000 |
| 0 | 19836 | 41049 | 78918 | 202841 | 418472 | 13585 | 22689 | 48841 | 131008 | 243069 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 1 | 13214 | 27624 | 53514 | 138098 | 279824 | 11791 | 25389 | 48102 | 123764 | 254777 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 2 | 15293 | 32291 | 62206 | 158877 | 324577 | 11793 | 25391 | 48106 | 123577 | 254777 | Optimal | Optimal | Optimal | – | – |
| 3 | 12691 | 27042 | 51707 | 132916 | 272157 | 14191 | 30342 | 58007 | 148516 | 304057 | Optimal | – | Optimal | – | – |
| 4 | 15308 | 32284 | 62209 | 159060 | 324609 | 11792 | 25391 | 48106 | 123690 | 254777 | Optimal | Optimal | Optimal | – | Optimal |
| 5 | 11793 | 25391 | 48106 | 123764 | 254777 | 11793 | 25391 | 48106 | 123764 | 254777 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 6 | 17920 | 37000 | 72399 | 187312 | 379545 | 2401186 | 4902351 | 9904984 | 24712830 | 49523374 | Optimal | Optimal | Optimal | – | Optimal |
| 7 | 11948 | 22147 | 43417 | 119279 | 232145 | 6578 | 11914 | 25921 | 63365 | 120497 | Optimal | – | – | – | – |
| 8 | 15276 | 27478 | 60080 | 145888 | 277084 | 6437 | 11951 | 23770 | 64611 | 125478 | Optimal | – | – | – | – |
| 9 | 27261 | 49204 | 104200 | 263953 | 504878 | 6461 | 12004 | 23700 | 64553 | 125478 | Optimal | – | – | – | – |
| 10 | 19390 | 40686 | 79202 | 201662 | 409472 | 11790 | 25386 | 48102 | 123762 | 254772 | Optimal | Optimal | Optimal | Optimal | – |
| 11 | 11784 | 25377 | 48087 | 123714 | 254682 | 11792 | 25390 | 48105 | 123763 | 254777 | Optimal | Optimal | – | – | – |
| 12 | 248488 | 535012 | 1013639 | 2607828 | 5368399 | 11793 | 25391 | 48106 | 123764 | 254777 | Optimal | – | – | – | – |

Thống kê đối với case s002.kp

| Instances | Values | | | | | Weights | | | | | Optimals | | | | |
|-----------|-----------------|-------|-------|--------|--------|-----------------|---------|---------|----------|----------|-----------------|---------|---------|---------|---------|
| | Number of items | | | | | Number of items | | | | | Number of items | | | | |
| | 50 | 100 | 200 | 500 | 1000 | 50 | 100 | 200 | 500 | 1000 | 50 | 100 | 200 | 500 | 1000 |
| 0 | 20561 | 45504 | 80297 | 201378 | 405157 | 13767 | 21604 | 51359 | 124347 | 247350 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 1 | 15065 | 29321 | 54414 | 138191 | 275159 | 13569 | 27380 | 48999 | 125404 | 249751 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 2 | 16970 | 34181 | 62992 | 160186 | 319409 | 13570 | 27381 | 48892 | 125186 | 249409 | Optimal | Optimal | – | – | – |
| 3 | 14346 | 28932 | 52700 | 133970 | 267457 | 16046 | 32332 | 58900 | 149770 | 299257 | Optimal | – | Optimal | – | – |
| 4 | 16983 | 34177 | 62989 | 160311 | 319499 | 13570 | 27381 | 48869 | 125306 | 249486 | Optimal | Optimal | – | – | – |
| 5 | 13570 | 27381 | 49000 | 125404 | 249752 | 13570 | 27381 | 49000 | 125404 | 249752 | Optimal | Optimal | Optimal | Optimal | Optimal |
| 6 | 19477 | 39395 | 74488 | 187186 | 372599 | 2401507 | 4902104 | 9904931 | 24712606 | 49525012 | Optimal | Optimal | Optimal | – | Optimal |

| | | | | | | | | | | | | | | | |
|----|--------|--------|---------|---------|---------|-------|-------|-------|--------|--------|---------|---------|---------|---|---|
| 7 | 15196 | 28162 | 55175 | 137722 | 275572 | 933 | 1672 | 3315 | 8271 | 16644 | Optimal | Optimal | – | – | – |
| 8 | 2480 | 4452 | 8902 | 22174 | 44592 | 13330 | 24264 | 47904 | 119464 | 239682 | Optimal | Optimal | Optimal | – | – |
| 9 | 28668 | 51730 | 103431 | 257814 | 517644 | 13268 | 23930 | 47831 | 119214 | 239344 | – | Optimal | – | – | – |
| 10 | 21166 | 42678 | 80096 | 201600 | 402650 | 13566 | 27378 | 48996 | 125400 | 249750 | Optimal | Optimal | Optimal | – | – |
| 11 | 13566 | 27372 | 48978 | 125352 | 249630 | 13570 | 27381 | 49000 | 125404 | 249751 | Optimal | Optimal | Optimal | – | – |
| 12 | 285931 | 576941 | 1032473 | 2642373 | 5262502 | 13570 | 27381 | 49000 | 125404 | 249752 | Optimal | – | – | – | – |

Chúng ta có thể rút ra một vài nhận xét sau khi đọc số liệu thống kê của 3 bảng trên

- ✓ Ở nhóm test case số 12 thì luôn có số lượng values lớn nhất so với các nhóm test case còn lại. Trong khi đó ở nhóm test case số 6 thì luôn có số lượng weight lớn nhất so với phần test case còn lại.
- ✓ Cả 3 nhóm test case số 0, 1 và 5 cả ba lần chạy thì đều đưa ra kết quả optimal cho nên ta có thể kết luận đây là nhóm test case dễ nhất. Ngoài ra, nhóm test case số 3 và số 10 cũng là nhóm test case dễ.
- ✓ Nhóm test case số 9 là test case ít chạy ra kết quả optimal nhất sau cả ba lượt chạy. Do đó ta có thể kết luận đây là nhóm test case khó nhất. Ngoài ra, nhóm test case số 7 và số 12 cũng là nhóm test case khó.

4. TỰ LIỆU THAM KHẢO

- 1) Solving knapsack with OR-Tools: <https://developers.google.com/optimization/bin/knapsack>
- 2) Knapsack test instances: <https://github.com/likr/kplib>
- 3) Branch and Bound: <https://viblo.asia/p/nhanh-va-can-branch-and-bound-Qbq5QBPEKD8>
- 4) Knapsack problems: https://en.wikipedia.org/wiki/Knapsack_problem

KẾT THÚC