Phân tích và thiết kế thuật toán
CS112.L11.KHCL
Bài toán Knapsack 0-1

18520430 18520456 18521630

1 Giới thiệu bài toán

1.1 Mô tả bài toán

Bài toán Knapsack (hay còn gọi là bài toán xếp balo) là một bài toán thuộc lĩnh vực tối ưu hóa tổ hợp. Cụ thể, bài toán đưa ra vấn đề đó là giữa những món đồ được cung cấp, phải chọn ra những món đồ nào để xếp vào balo có giới hạn khối lượng chứa sao cho tối ưu một tiêu chí nào đó (ví dụ như giá trị, nhu cầu sử dụng, ...).
Bài toán xếp balo là một bài toán kinh điển, đã được nghiên cứu trong hơn một thế kỷ, các công trình nghiên cứu sớm nhất về bài toán này đã có từ năm 1897. Tên gốc Tiếng Anh "Knapsack problem" chính thức được dùng để nói đến vấn đề này xuất phát từ công trình nghiên cứu đầu tiên của nhà toán học Tobias Dantzig (1884 - 1956), ý tưởng được đề cập đến đó là đóng gói những vật dụng có giá trị và hữu ích nhất vào một chiếc balo có giới hạn về sức chứa sao cho không làm balo bị quá tải.
Bài toán xếp balo dạng 0-1 là một phần trong bài toán xếp balo, món đồ tương ứng với 1 là món đồ được chọn, tương ứng với 0 là không được chọn.
Giả sử ta có n đồ vật: $x_1, ..., x_n$, mỗi đồ vật i có giá trị v_i và khối lượng w_i, khối lượng tối đa mà có thể mang vào ba lô là C, bài toán xếp balo 0-1 có thể được phát biểu như sau:

$$\max \sum_{i=1}^{n} v_i x_i$$

sao cho:

$$\sum_{i=1}^{n} w_i x_i \leq C,\ \mathrm{x}_i = 0|1$$

1.2 Phát biểu bài toán

Bạn Huỳnh Anh sắp tới phải thi cuối kỳ môn Phân tích thiết kế thuật toán, để ôn tập cũng như là trau dồi thêm một số kiến thức mới, bạn Huỳnh Anh quyết định học thêm một số khóa học online.

Tuy nhiên, các khóa học này cần phải trả phí mới có thể học được và hiện tại trong tài khoản cá nhân của bạn chỉ còn 30 đô la nếu quy đổi sang tiền đô. Mỗi khóa học trên trang web sẽ cung cấp thông về giá tiền và số chương sẽ giảng dạy. Bạn Huỳnh Anh mong muốn với học được nhiều chương nhất trong phạm vi tài chính mà bạn có thể chi trả được.
Hãy giúp bạn Huỳnh Anh có thể học được số chương nhiều nhất với số tiền hiện có.
INPUT: Thông tin các khóa học, mỗi khóa học cung cấp thông tin về giá tiền và số chương giảng dạy.
OUTPUT: Số chương tối đa mà bạn Huỳnh Anh có thể học được với số tiền có trong tài khoản.

2 Ý tưởng thiết kế thuật toán

2.1 Bottom-up Dynamic programming

Phương pháp quy hoạch động (tiếng Anh: Dynamic Programming, viết tắt: DP) được thiết kế bởi nhà toán học người Mỹ - Richard Bellmam vào năm 1953 và được áp dụng rộng rãi trong nhiều lĩnh vực, ví dụ như Kỹ thuật hàng không vũ trụ, kinh tế,... Đây là một phương pháp tối ưu hóa trong toán học và lập trình. Trong Khoa học Máy tính nói chung và lập trình nói riêng, quy hoạch động được thiết kế để tối ưu bài toán đệ quy thông thường, giảm thời gian thực thi chương trình thể hiện các tính chất của các bài toán con gối nhau (tiếng Anh: overlapping subproblem) và cấu trúc con tối ưu (tiếng Anh: optimal substructure). Nguyên lý "chia để trị" thường đóng vai trò chủ đạo trong thiết kế thuật toán đặc biệt là phương pháp quy hoạch đông. Ý tưởng chính là đơn giản hóa bài toán lớn thành một số hữu hạn bài toán nhỏ hơn và lưu trữ kết quả tối ưu của các bài tóan nhỏ này lại, do đó không cần phải tính lại các bước thực hiện trước đó - đây là điểm cải tiến của quy hoạch động so với đệ quy thông thường. Phương pháp quy hoạch động sử dụng:

- Các bài toán con gối nhau: là cùng các bài toán nhỏ hơn bị lặp lại và giải quyết nhiều lần.
- Cấu trúc con tối ưu: là các lời giải tối ưu cho các bài toán con có thể được sử dụng để tìm các lời giải tối ưu cho bài toán toàn cục.
- Memoization: là kỹ thuật lưu trữ lại các kết quả của các bài toán con và trả về giá trị đã lưu nếu bài toán đó đã được giả quyết.

Có 2 hướng tiếp cận chính trong phương pháp quy hoạch động, Top-down và Bottom-up.

- Phép phân rã đệ quy theo hướng Top-down tức là bắt đầu từ bài toán lớn phân rã thành nhiều bài toán con và đi giải quyết từng bài toán con đó. Việc phân rã từng bài toán con lại đưa về phép phân rã tiếp thành nhiều bài toán nhỏ hơn và lại tiếp tục đi giải tiếp những bài toán con đó bất kể

nó đã được giải hay chưa. Do đó thuật giải đệ quy trong quy hoạch động tốn nhiều thời gian và không gian lưu trữ hơn hướng tiếp cận Bottom-up.

- Quy hoạch động theo hướng Bottom-up có nghĩa là sẽ bắt đầu tính toán tất cả các bài toán nhỏ (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn cho tới khi giải quyết được bài toán lớn nhất (bài toán ban đầu).

Để giải quyết bài toán Knapsack 0-1 một cách tối ưu nhất, nhóm sẽ sử dụng phương pháp tiếp cận là Bottom-up để tối ưu hóa thời gian do không cần phải thực hiện lời gọi đệ quy. Chỉ cần tính các bài toán con, lưu và sử dụng lại trong bảng tra và sẽ trả về kết quả khi toàn bộ giá trị trong bảng tra được tính.
Công thức đệ qui của Knapsack 0-1:

$$R[i,w] = \begin{cases} R[i-1,w], & \text{nếu } w_i > w. \\ \max(R[i-1,w-w_i]+v_i, R[i-1,w]), & \text{nếu } w_i \leq w. \end{cases} \quad (1)$$

2.2 Mã giả

Algorithm 1 Bottom-Up Dynamic programming cho bài toán Knapsack 0-1

```
Khởi tạo ma trận 2 chiều R với kích thước n*C
for i = 1, ..., n do
    for w = 1, ..., C do
        if w_i ≤ w then
            R[i, w] = max(R[i − 1, w − w_i] + v_i, R[i − 1, w])
        else
            R[i, w] = R[i − 1, w]
        end if
    end for
end for
return R[n, C]
```

Algorithm 2 Knapsack 0-1 tối ưu bộ nhớ

```
Khởi tạo ma trận 1 chiều R với kích thước W
for i = 1, ..., n do
    for j = C, ..., w do
        R[j] = max(R[j], v_i + R[j − w_i])
    end for
end for
return R[n, C]
```

Để tối ưu bộ nhớ, thay vì sử dụng một ma trận 2 chiều như mã giả miêu tả lúc đầu để lưu kết quả của những bài toán con thì chỉ cần dử dụng một ma

trận một chiều. Kết quả của bài toán vẫn nằm ở vị trí cuối cùng của ma trận. Vì vậy, chỉ vần thay thế các giá trị tại những vị trí j (ở mã giả Algorithm 2) mỗi khi một vòng for mới thực hiện. Với cách thực hiện này, so với ban đầu lưu n * C phần tử, thì chỉ cần lưu C phần tử.

Algorithm 3 Trace back Knapsack 0-1

```
1: Khởi tạo ma trận 2 chiều R với kích thước n*C
2: for i = 1, ..., n do
3:     for w = 1, ..., C do
4:         if w_i ≤ w then
5:             R[i, w] = max(R[i − 1, w − w_i] + v_i, R[i − 1, w])
6:         else
7:             R[i, w] = R[i − 1, w]
8:         end if
9:     end for
10: end for
11: Result = R[n, C]
12: P = C
13: for i = n, ..., 0 do
14:     if Result ≤ 0 then
15:         break
16:     end if
17:     if Result == R[i − 1, C] then
18:         continue
19:     else
20:         print w[i − 1]
21:         Result = Result − v[i − 1]
22:         P = P − w[i − 1]
23:     end if
24: end for
```

Traceback thật ra là bản chất vẫn là phương pháp đệ quy thông thường, thuật toán chính giống với mã giả mô tả ở Algorithm 1, tuy nhiên Traceback có thêm một phần đó là in ra giá trị khối lượng các vật thể được chọn. Cụ thể, thuật toán sẽ xét phần tử cuối cùng trước, phần tử R[n,C], nếu như phần tử này bằng phần tử ở vị trí này ở hàng trên (tức R[n-1,C]), có nghĩa là phần tử này không ảnh hưởng đến giá trị cuối cũng, ngược lại đây là phần tử được chọn. Nếu như phần tử ở vị trí hàng đó được chọn, thuật toán sẽ duyệt hàng đó từ phải sang trái, đến khi gặp một phần tử nào đó không thay đổi so với hàng trên thì nhảy đến hàng trên để tiếp tục quá trình tương tự.

2.3 Phân tích độ phức tạp bằng toán học

Với mỗi lần thực thi, thuật toán phải duyệt qua toàn bộ giá trị w cho đến C và chọn từng vật thể i để tính giá trị mới của kết quả.

$$T(C) = \sum_{i=1}^{n} \sum_{j=1}^{C} c$$

Ta có:

$$T(C) = \sum_{i=1}^{n} \sum_{j=1}^{C} c = \sum_{i=1}^{n} Cc = nCc$$

Vậy:

$$T(C) \subset O(nC)$$

Kết luận: Độ phức tạp của thuật toán là O(nC), với n là số lượng vật thể và C là giá trị giới hạn của vật chứa.

2.4 Mã nguồn - Python

Knapsack 01 (Chưa tối ưu bộ nhớ):

```
def KnapSack(C, wt, val, n):
  R = [[0 for x in range(C + 1)] for x in range(n + 1)]
  for i in range(n + 1):
    for w in range(C + 1):
      if i == 0 or w == 0:
        R[i][w] = 0
      elif wt[i-1] <= w:
        R[i][w] = max(val[i-1]+R[i-1][w-wt[i-1]], R[i-1][w])
      else:
        R[i][w] = R[i-1][w]
  return R[n][C]
```

Knapsack tối ưu bộ nhớ:

```
def KnapSack(C, wt, val, n):
  R = [0]*(C+1)
  for i in range(n):
    for j in range(C,wt[i],-1):
      R[j] = max(R[j] , val[i] + R[j-wt[i]])
  return R
```

Knapsack 01 có trace back:

```
def TraceBack(C, wt, val, n):
  R = [[0 for x in range(C + 1)] for x in range(n + 1)]
  for i in range(n + 1):
    for w in range(C + 1):
      if i == 0 or w == 0:
        R[i][w] = 0
      elif wt[i-1] <= w:
        R[i][w] = max(val[i-1] + R[i-1][w-wt[i-1]], R[i-1][w])
      else:
        R[i][w] = R[i-1][w]
```

```
  result = R[n][C]
  print(result)
  w = C
  for i in range(n, 0, -1):
    if result <= 0:
        break
    if result == R[i - 1][w]:
        continue
    else:
        print(wt[i - 1])
        result = result - val[i - 1]
        w = w - wt[i - 1]
```

2.5 Phát sinh Input - Output và kiểm tra tính đúng

Sử dụng các hàm Random để phát sinh ngẫu nhiên các thông tin cho phần input như: số lượng phần tử, giá trị các phần tử trong mảng V, W và sức chứa C. Để phát sinh ouput, xây dựng chương trình bằng công thức đệ quy, sau đó với các input vừa tạo ra sẽ tìm ra được output tương ứng.
Để kiểm tra tính đúng đắn của thuật toán quy hoạch động trong bài toán này, cũng với các input tạo ra được, lấy được output tương ứng, sau đó lấy kết quả này so sánh với kết quả chạy thực nghiệm với chương trình sử dụng công thức đệ quy.

2.6 Phân tích độ phức tạp thời gian bằng thực nghiệm

Môi trường thực hiện: **Google Colab**.
CPU: **Intel(R) Xeon(R) 2.30GHz**.
RAM: **12GB**.
Chương trình dừng lại khi n = **1000**, C = **663429**.
Sử dụng Linear Regression để dự đoán độ phức tạp thời gian của Knapsack 0-1.
Công thức Linear Regression:

$$T(C) = w_0 f(C) + b$$

Với $f(C) \in \{log_2(C), sqrt(C), C, C^2, C^3\}$, w_0 trọng số và b là sai số, Mean square error (MSE) là độ lỗi. Kết quả thực nghiệm được ghi nhận trong bảng sau:

	$log(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
w_0	42.03875856	0.46022488	0.00067094	3.61E-05	1.75E-09	4.40E-15
b	-600.5307118	-64.0087777	-3.040067618	1.810506485	28.90207545	40.75206171

Bảng 1: Giá trị của w_0 và b cho từng trường hợp của $f(C)$.

C	$t(C)$	$log(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
10000	4.275	-41.93177927	-17.9862897	3.669332382	2.980368168	29.07659961	40.75646144
10700	4.716	-37.82834256	-16.40274595	4.138990382	3.353386438	29.10188816	40.75745157
11449	5.193	-33.72490584	-14.76471554	4.641524442	3.755151979	29.13084103	40.75866452
12250	5.761	-29.623598	-13.07121772	5.178947382	4.187628707	29.16397077	40.76014958
13108	6.072	-25.51784781	-11.317551	5.754613902	4.653900639	29.20194228	40.76197083
14026	6.942	-21.41250848	-9.503694063	6.370536822	5.15601142	29.24541452	40.76420195
15007	7.423	-17.31238562	-7.629821071	7.028728962	5.696032622	29.2951214	40.76693159
16058	8.076	-13.20702267	-5.689005369	7.733886902	6.278285712	29.35210234	40.77027969
17182	8.623	-9.103797744	-3.682441449	8.488023462	6.904937292	29.41730764	40.77437928
18385	9.573	-4.999503328	-1.606289012	9.295164282	7.579867011	29.49198147	40.77940285
19672	10.206	-0.895912465	0.540940552	10.15866406	8.306453946	29.57746229	40.78555604
21049	11.341	3.207408994	2.761906608	11.08254844	9.08869732	29.6753229	40.79309347
22522	12.451	7.309687778	5.058695874	12.07084306	9.930659363	29.78733265	40.8023245
24098	13.531	11.41176545	7.434372031	13.1282445	10.83704067	29.915561	40.81363165
25785	14.452	15.51553146	9.892791645	14.26012028	11.81319616	30.06242765	40.82748877
27590	15.467	19.61907808	12.43566931	15.47116698	12.86398323	30.2305675	40.8444636
29522	16.471	23.72395287	15.06691064	16.76742306	13.99550448	30.42313813	40.86526605
31588	17.679	27.82635279	17.7870512	18.1535851	15.21277727	30.64348057	40.89073478
33799	19.252	31.9295024	20.60128535	19.63703344	16.52326309	30.8957913	40.92193977
36165	20.724	36.0330565	23.51263677	21.22447748	17.93394586	31.18468956	40.96017084
38697	22.458	40.13719849	26.52460378	22.92329756	19.4525149	31.51550113	41.00701324
41406	24.036	44.2409428	29.63991777	24.74087402	21.08677491	31.89421686	41.06439305
44304	26.078	48.34380457	32.86172698	26.68525814	22.84525178	32.3277132	41.13467015
47405	28.385	52.44688305	36.19455557	28.76584308	24.73781925	32.82404178	41.2207652
50724	30.046	56.55109695	39.64302032	30.99269294	26.77511539	33.39244978	41.32626571
54274	32.759	60.6537738	43.20880421	33.37452994	28.96669953	34.0429762	41.45545986
58074	35.441	64.75806688	46.89873441	35.92410194	31.32599433	34.78805894	41.61379119
62139	38.625	68.8613279	50.71467873	38.65147304	33.8641262	35.6408989	41.8077082
66488	42.404	72.96409873	54.66143692	41.5693911	36.59488795	36.61718501	42.04523146
71143	44.635	77.06825154	58.74537149	44.6926168	39.53417157	37.73531297	42.33630637
76123	47.388	81.17168029	62.96909323	48.033898	42.69619618	39.01524643	42.69282878
81451	50.466	85.27466279	67.33766147	51.60866632	46.09793156	40.48047148	43.12953141
87153	54.447	89.37839873	71.85737061	55.4343662	49.75851941	42.15831187	43.66460743
93253	58.423	93.48137368	76.5317239	59.5271002	53.69608411	44.07890943	44.31997396
99781	62.616	97.58498667	81.36765882	63.90699652	57.93289741	46.27813367	45.12294575
106766	70.491	101.6886108	86.37000994	68.59351242	62.49089734	48.79604749	46.10663225
114239	74.102	105.7917184	91.54382767	73.60744704	67.39363697	51.67843681	47.31153407
122236	79.801	109.8952891	96.89626615	78.97295422	72.66829587	54.97884677	48.78775066
130793	84.751	113.9989484	102.4329993	84.7141878	78.34243579	58.75759007	50.59623458
139948	89.557	118.1021641	108.1596214	90.8566435	84.44532867	63.08340501	52.81146494
149745	96.499	122.20586	114.0839797	97.42984268	91.01066993	68.03661418	55.52553864
160227	107.931	126.3092399	120.2117168	104.4626358	98.07194637	73.70712495	58.85015538

171443	110.541	130.4127155	126.5504447	111.9878988	105.6671603	80.19944245	62.92304735
183444	120.594	134.5161489	133.1072093	120.0398497	113.8361938	87.63242452	67.91246802
196285	128.091	138.6195609	139.8895432	128.6553903	122.6222038	96.14239729	74.02468666
210025	136.199	142.7230121	146.9053105	137.8741059	132.0716793	105.8855566	81.5124871
224726	146.412	146.8262464	154.1620875	147.7375948	142.2338046	117.0398747	90.68483558
240457	157.544	150.9297285	161.6690046	158.292152	153.1632834	129.8111929	101.9219942
257289	173.284	155.0331676	169.43414	169.585414	164.9169457	144.432933	115.687868
275299	186.425	159.1365536	177.4663532	181.6690434	177.5565798	161.1731332	132.5514166
294570	196.423	163.2400047	185.7750696	194.5987282	191.14902	180.3392815	153.210299
315190	205.746	167.3434607	194.369684	208.433511	205.7655215	202.2826426	178.5183692
337253	222.259	171.4468435	203.2598624	223.2364602	221.4825416	227.4051337	209.521262
360861	241.848	175.5503289	212.4561644	239.0760117	238.3832582	256.1685921	247.5020876
386122	252.174	179.6538803	221.9690525	256.0246271	256.556244	289.1004942	294.0311702
413150	274.009	183.7572377	231.8087967	274.1587934	276.0955465	326.8024663	351.028644

Bảng 2: Thời gian thực thi của thuật toán $T(C)$ được tính bằng công thức $w_0 f(C) + b$ với các giá trị C, mỗi giá trị C bằng 1.07 lần so với giá trị C trước đó.

	$log_2(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
MSE	970.1571	227.1509	1.9568	19.3431	450.3811	1126.4100

Bảng 3: MSE tương ứng với từng $f(n)$.

Từ kết quả trên, có thể nhận thấy rằng $f(C) = C$ cho độ lỗi MSE nhỏ nhất. Do số lượng phần tử **n** được giữ nguyên với **n=1000** và **T(C)** tuyến tính theo C cho nên độ phức tạp của thuật toán Knapsack 0-1 là **O(nC)**.

2.7 Phân tích độ phức tạp bộ nhớ bằng thực nghiệm (Chưa tối ưu bộ nhớ)

Môi trường thực hiện: **Google Colab**.
CPU: **Intel(R) Xeon(R) 2.30GHz**.
RAM: **12GB**.
Sử dụng Linear Regression để dự đoán độ phức tạp của Knapsack 0-1. Công thức Linear Regression:

$$S(C) = w_0 f(C) + b$$

Với $f(C) \in \{log_2(C), sqrt(C), C, C^2, C^3\}$, w_0 trọng số và b là sai số, Mean square error (MSE) là độ lỗi. Kết quả thực nghiệm được ghi nhận trong bảng sau:

	$log(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
w_0	95.67994482	2.33414622	0.00763702	0.0004712	1.75E-09	1.81E-12
b	-1151.255244	-151.3371427	0.007637024	14.80914234	28.90207545	129.0876137

Bảng 4: Giá trị của w_0 và b cho từng trường hợp của $f(C)$.

C	$S(C)$	$log(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
1000	7.64466095	-197.7295538	-77.52495819	7.644657024	19.50501989	28.90382069	129.0894259
2000	15.28168488	-102.049609	-46.95095032	15.28167702	25.14329745	28.90905642	129.1021114
3000	22.9187088	-46.0804292	-23.49068894	22.91869702	31.13727799	28.91778263	129.1365434
4000	30.55573273	-6.369664166	-3.712773721	30.55571702	37.36225256	28.92999932	129.2035952
5000	38.19275665	24.43239819	13.71191938	38.19273702	43.7589927	28.94570649	129.31414
6000	45.82978058	49.59951562	29.46504609	45.82975702	50.29261365	28.96490415	129.4790511
7000	53.4668045	70.87801023	43.95154117	53.46677702	56.9400647	28.98759229	129.7092019
8000	61.10382843	89.31028065	57.43524203	61.10379702	63.68496278	29.01377091	130.0154654
9000	68.74085236	105.5686954	70.09941075	68.74081702	70.51505828	29.04344002	130.4087151
10000	76.37787628	120.112343	82.07747934	76.37783702	77.42084307	29.07659961	130.8998241
11000	84.01490021	133.2686726	93.47017818	84.01485702	84.39472141	29.11324969	131.4996657
12000	91.65192413	145.2794604	104.3557648	91.65187702	91.43048496	29.15339024	132.2191133
13000	99.28894806	156.3283142	114.7964732	99.28889702	98.52296409	29.19702128	133.0690399
14000	106.925972	166.557955	124.8427626	106.925917	105.6677871	29.24414281	134.060319
15000	114.5629959	176.0815228	134.5362185	114.562937	112.8612084	29.29475481	135.2038238
16000	122.2000198	184.9902255	143.9115952	122.199957	120.0999832	29.3488573	136.5104275
17000	129.8370438	193.3586653	152.9982904	129.836977	127.3812731	29.40645028	137.9910034
18000	137.4740677	201.2486402	161.8214344	137.473997	134.7025742	29.46753373	139.6564247
19000	145.1110916	208.7119163	170.4027102	145.111017	142.0616613	29.53210767	141.5175648
20000	152.7481155	215.7922878	178.7609814	152.748037	149.4565438	29.6001721	143.5852968
21000	160.3851395	222.5271348	186.9127799	160.385057	156.8854304	29.671727	145.8704941
22000	168.0221634	228.9486174	194.8726906	168.022077	164.3467005	29.74677239	148.3840299
23000	175.6591873	235.0846045	202.6536584	175.659097	171.8388812	29.82530826	151.1367775
24000	183.2962112	240.9594053	210.2672348	183.296117	179.3606276	29.90733462	154.1396101
25000	190.9332352	246.5943502	217.7237797	190.933137	186.9107071	29.99285146	157.403401
26000	198.5702591	252.0082591	225.0326263	198.570157	194.4879858	30.08185878	160.9390235
27000	206.207283	257.21782	232.2022185	206.207177	202.0914171	30.17435659	164.7573508
28000	213.8443069	262.2378999	239.240225	213.844197	209.7200318	30.27034487	168.8692562
29000	221.4813309	267.0817997	246.1536355	221.481217	217.3729301	30.36982365	173.2856129
30000	229.1183548	271.7614676	252.9488419	229.118237	225.0492744	30.4727929	178.0172942
31000	236.7553787	276.2876758	259.6317076	236.755257	232.7482829	30.57925264	183.0751734
32000	244.3924026	280.6701703	266.2076267	244.392277	240.4692241	30.68920286	188.4701238
33000	252.0294266	284.9177972	272.6815751	252.029297	248.2114124	30.80264356	194.2130185
34000	259.6664505	289.0386101	279.0581542	259.666317	255.9742039	30.91957475	200.3147309
35000	267.3034744	293.0399622	285.3416296	267.303337	263.7569923	31.03999642	206.7861342
36000	274.9404984	296.9285851	291.5359642	274.940357	271.5592061	31.16390858	213.6381017

37000	282.5775223	300.7106568	297.6448473	282.577377	279.3803056	31.29131122	220.8815066
38000	290.2145462	304.3918611	303.6717209	290.214397	287.2197803	31.42220434	228.5272222
39000	297.8515701	307.9774388	309.6198016	297.851417	295.0771465	31.55658794	236.5861218
40000	305.4885941	311.4722326	315.4921013	305.488437	302.9519453	31.69446203	245.0690787
41000	313.125618	314.8807264	321.2914447	313.125457	310.8437408	31.8358266	253.986966
42000	320.7626419	318.2070797	327.0204853	320.762477	318.7521184	31.98068165	263.3506571
43000	328.3996658	321.4551585	332.6817193	328.399497	326.6766832	32.12902719	273.1710252
44000	336.0366898	324.6285622	338.277499	336.036517	334.6170586	32.28086321	283.4589436
45000	343.6737137	327.7306474	343.8100435	343.673537	342.5728854	32.43618971	294.2252855
46000	351.3107376	330.7645494	349.2814492	351.310557	350.54382	32.5950067	305.4809242
47000	358.9477615	333.733201	354.6936994	358.947577	358.529534	32.75731417	317.236733
48000	366.5847855	336.6393501	360.0486722	366.584597	366.5297128	32.92311212	329.5035852
49000	374.2218094	339.4855743	365.3481486	374.221617	374.5440552	33.09240056	342.2923539
50000	381.8588333	342.274295	370.5938191	381.858637	382.5722719	33.26517948	355.6139125
51000	389.4958572	345.0077899	375.7872899	389.495657	390.6140855	33.44144888	369.4791341
52000	397.1328812	347.6882039	380.9300891	397.132677	398.6692293	33.62120876	383.8988922
53000	404.7699051	350.3175592	386.0236715	404.769697	406.737447	33.80445913	398.8840599
54000	412.406929	352.8977648	391.0694236	412.406717	414.8184918	33.99119999	414.4455106
55000	420.0439529	355.4306246	396.0686681	420.043737	422.9121262	34.18143132	430.5941173
56000	427.6809769	357.9178447	401.0226679	427.680757	431.0181212	34.37515314	447.3407536
57000	435.3180008	360.3610409	405.9326294	435.317777	439.1362561	34.57236544	464.6962925
58000	442.9550247	362.7617446	410.7997067	442.954797	447.2663179	34.77306823	482.6716073
59000	450.5920486	365.1214086	415.6250042	450.591817	455.408101	34.9772615	501.2775714
60000	458.2290726	367.4414124	420.4095797	458.228837	463.5614066	35.18494525	520.5250579
61000	465.8660965	369.7230674	425.1544472	465.865857	471.7260426	35.39611948	540.4249402
62000	473.5031204	371.9676206	429.8605791	473.502877	479.9018234	35.6107842	560.9880915

Bảng 5: Bộ nhớ lưu trữ của thuật toán $S(C)$ được tính bằng công thức $w_0 f(C)+b$ với các giá trị C, mỗi giá trị C tăng thêm 1000 đơn vị so với giá trị C trước đó.

	$log_2(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
MSE	3399.301115	653.0471008	2.02E-08	76554.10092	61985.08315	2930.604781

Bảng 6: MSE tương ứng với từng $f(n)$.

Từ kết quả trên, có thể nhận thấy rằng $f(C) = C$ cho độ lỗi MSE nhỏ nhất. Do số lượng phần tử n được giữ nguyên với **n=100**0 và **S(C)** tuyến tính theo C cho nên độ phức tạp bộ nhớ của thuật toán Knapsack 0-1 là **O(nC)**.

2.8 Phân tích độ phức tạp bộ nhớ bằng thực nghiệm (Có tối ưu bộ nhớ)

Môi trường thực hiện: **Google Colab**.
CPU: **Intel(R) Xeon(R) 2.30GHz**.
RAM: **12GB**.
Sử dụng Linear Regression để dự đoán độ phức tạp của Knapsack 0-1. Công thức Linear Regression:

$$S(C) = w_0 f(C) + b$$

Với $f(C) \in \{log_2(C), sqrt(C), C, C^2, C^3\}$, w_0 trọng số và b là sai số, Mean square error (MSE) là độ lỗi. Kết quả thực nghiệm được ghi nhận trong bảng sau:

	$log(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
w_0	0.09552	0.00233114	7.6288e-06	4.70703e-07	1.75E-09	1.81060e-15
b	1.13775e-10	-0.151441	-0.0003433	0.0144355	0.09063377	0.1285767

Bảng 7: Giá trị của w_0 và b cho từng trường hợp của $f(C)$.

C	$S(C)$	$log(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
1000	0.007637024	-0.197636503	-0.077724549	0.007285439	0.01912652	0.090747546	0.128578563
2000	0.015266418	-0.102109913	-0.047189919	0.014914257	0.024758856	0.091088874	0.128591238
3000	0.022895813	-0.04623044	-0.023759873	0.022543075	0.03074652	0.091657752	0.128625639
4000	0.030525208	-0.006583323	-0.00400743	0.030171893	0.036964934	0.092454182	0.128692632
5000	0.038154602	0.024169371	0.013394821	0.037800712	0.043354934	0.093478163	0.128803079
6000	0.045783997	0.04929615	0.029127659	0.04542953	0.049881669	0.094729696	0.128967844
7000	0.053413391	0.07054054	0.043595497	0.053058348	0.056522115	0.096208781	0.129197791
8000	0.061042786	0.088943267	0.057061831	0.060687166	0.063259905	0.097915416	0.129503784
9000	0.06867218	0.105175623	0.069709689	0.068315984	0.070082803	0.099849603	0.129896685
10000	0.076301575	0.119695961	0.081672331	0.075944802	0.07698131	0.102011342	0.13038736
11000	0.083930969	0.132831203	0.093050357	0.083573621	0.08394784	0.104400632	0.130986671
12000	0.091560364	0.14482274	0.103921924	0.091202439	0.090976189	0.107017473	0.131705482
13000	0.099189758	0.155853885	0.114349185	0.098831257	0.098061194	0.109861866	0.132554656
14000	0.106819153	0.16606713	0.124382536	0.106460075	0.105198487	0.11293381	0.133545058
15000	0.114448547	0.175575434	0.134063507	0.114088893	0.112384328	0.116233306	0.134687552
16000	0.122077942	0.184469857	0.143426809	0.121717711	0.119615475	0.119760353	0.135992999
17000	0.129707336	0.192824884	0.152501801	0.129346529	0.126889091	0.123514951	0.137472265
18000	0.137336731	0.200702213	0.161313581	0.136975348	0.134202677	0.127497101	0.139136213
19000	0.137336731	0.208153527	0.169883805	0.144604166	0.141554009	0.131706803	0.140995706
20000	0.15259552	0.215222551	0.178231312	0.152232984	0.148941099	0.136144055	0.143061609
21000	0.160224915	0.221946603	0.186372611	0.159861802	0.156362157	0.14080886	0.145344785
22000	0.167854309	0.228357793	0.19432227	0.16749062	0.163815564	0.145701215	0.147856097

23000	0.167854309	0.234483946	0.202093216	0.175119438	0.171299849	0.150821122	0.150606409
24000	0.183113098	0.24034933	0.209696987	0.182748257	0.178813669	0.156168581	0.153606585
25000	0.190742493	0.245975244	0.217143928	0.190377075	0.186355792	0.161743591	0.156867488
26000	0.198371887	0.251380475	0.224443362	0.198005893	0.193925086	0.167546152	0.160399982
27000	0.206001282	0.256581686	0.23160372	0.205634711	0.201520504	0.173576265	0.164214931
28000	0.213630676	0.26159372	0.238632662	0.213263529	0.20914108	0.179833929	0.168323199
29000	0.221260071	0.266429856	0.245537168	0.220892347	0.216785913	0.186319144	0.172735648
30000	0.228889465	0.271102024	0.252323623	0.228521166	0.224454168	0.193031911	0.177463143
31000	0.23651886	0.275620977	0.258997882	0.236149984	0.232145063	0.19997223	0.182516547
32000	0.244148254	0.279996447	0.265565332	0.243778802	0.239857868	0.2071401	0.187906725
33000	0.251777649	0.284237266	0.272030942	0.25140762	0.247591898	0.214535521	0.193644538
34000	0.259407043	0.288351474	0.278399308	0.259036438	0.255346509	0.222158494	0.199740852
35000	0.267036438	0.292346413	0.284674691	0.266665256	0.263121095	0.230009018	0.20620653
36000	0.274665833	0.296228803	0.290861048	0.274294075	0.270915087	0.238087093	0.213052435
37000	0.282295227	0.300004813	0.296962063	0.281922893	0.278727945	0.24639272	0.220289432
38000	0.289924622	0.303680117	0.302981175	0.289551711	0.286559158	0.254925899	0.227928383
39000	0.297554016	0.307259948	0.308921595	0.297180529	0.294408244	0.263686629	0.235980152
40000	0.305183411	0.310749141	0.314786331	0.304809347	0.302274744	0.27267491	0.244455604
41000	0.312812805	0.314152171	0.320578205	0.312438165	0.310158223	0.281890743	0.253365601
42000	0.3204422	0.317473193	0.326299867	0.320066983	0.318058267	0.291334127	0.262721008
43000	0.328071594	0.320716066	0.33195381	0.327695802	0.32597448	0.301005062	0.272532687
44000	0.335700989	0.323884383	0.337542383	0.33532462	0.333906488	0.310903549	0.282811504
45000	0.343330383	0.326981497	0.343067802	0.342953438	0.341853931	0.321029587	0.29356832
46000	0.350959778	0.330010536	0.348532161	0.350582256	0.349816465	0.331383177	0.304814001
47000	0.358589172	0.332974429	0.35393744	0.358211074	0.357793764	0.341964319	0.316559408
48000	0.366218567	0.33587592	0.359285516	0.365839892	0.365785512	0.352773011	0.328815408
49000	0.373847961	0.338717583	0.364578167	0.373468711	0.373791409	0.363809255	0.341592862
50000	0.381477356	0.341501834	0.369817082	0.381097529	0.381811165	0.375073051	0.354902634
51000	0.38910675	0.344230947	0.375003864	0.388726347	0.389844504	0.386564398	0.368755589
52000	0.396736145	0.346907065	0.380140039	0.396355165	0.397891159	0.398283296	0.383162589
53000	0.40436554	0.349532206	0.385227061	0.403983983	0.405950875	0.410229746	0.398134499
54000	0.40436554	0.352108276	0.390266315	0.411612801	0.414023404	0.422403747	0.413682181
55000	0.419624329	0.354637076	0.395259121	0.41924162	0.422108509	0.434805299	0.429816501
56000	0.427253723	0.35712031	0.40020674	0.426870438	0.430205961	0.447434404	0.44654832
57000	0.434883118	0.35955959	0.405110378	0.434499256	0.438315541	0.460291059	0.463888504
58000	0.442512512	0.361956446	0.409971187	0.442128074	0.446437036	0.473375266	0.481847915
59000	0.450141907	0.364312328	0.41479027	0.449756892	0.454570239	0.486687024	0.500437418
60000	0.457771301	0.366628614	0.419568683	0.45738571	0.462714952	0.500226334	0.519667876
61000	0.465400696	0.368906612	0.424307439	0.465014529	0.470870984	0.513993195	0.539550152
62000	0.47303009	0.371147567	0.42900751	0.472643347	0.479038149	0.527987608	0.56009511

Bảng 8: Bộ nhớ lưu trữ của thuật toán $S(C)$ được tính bằng công thức $w_0 f(C)+b$ với các giá trị C, mỗi giá trị C tăng thêm 1000 đơn vị so với giá trị C trước đó.

	$log_2(C)$	$sqrt(C)$	C	$Clog(C)$	C^2	C^3
MSE	0.003410804	0.000662031	2.68E-06	1.69E-05	0.001125672	0.002921021

Bảng 9: MSE tương ứng với từng $f(n)$.

Từ bảng kết quả trên, có thể thấy rằng, với chương trình đã được tối ưu bộ nhớ, cùng số **C** nhưng độ phức tạp về mặt không gian lưu trữ nhỏ hơn **n** lần so với khi thực nghiệm phương pháp quy hoạch động Bottom-up thông thường, có thể kết luận rằng, với chương trình đã được tối ưu về mặt bộ nhớ, độ phức tạp không gian lưu trữ là **O(C)**.