Đại học Quốc gia Hà Nội
Trường Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học

Bài tập lớn môn học

THIẾT KẾ VÀ ĐÁNH GIÁ THUẬT TOÁN

Đề tài:

MỘT SỐ PHƯƠNG PHÁP GIẢI BÀI TOÁN XẾP BA LÔ

Sinh viên thực hiện (Nhóm 05):
Đặng Quang Vinh - 18001218
Ngô Quang Vinh - 18001219

Giảng viên hướng dẫn:
PGS.TS. Nguyễn Thị Hồng Minh

Hà Nội - 2021

Lời nói đầu

Thiết kế và đánh giá thuật toán là một trong những môn học nòng cốt, được coi là nền móng cơ sở, là bộ môn bắt buộc đối với sinh viên các ngành Khoa học máy tính, Công nghệ thông tin...

Cùng với Cấu trúc dữ liệu và giải thuật, Thiết kế và đánh giá thuật toán mang lại cho sinh viên những kiến thức cơ bản nhất về thuật toán, thiết kế thuật toán, đánh giá và triển khai thuật toán để giải quyết các vấn đề trong thực tế nói chung và các vấn đề chuyên sâu trong khoa học máy tính nói riêng.

Bài toán xếp ba lô là một bài toán kinh điển mà chắc chắn ai cũng từng phải gặp qua khi bước đầu tìm hiểu về phương pháp Quy hoạch động. Tuy nhiên, ngoài phương pháp giải phổ biến sử dụng Quy hoạch động ta có rất nhiều hướng tiếp cận khác cũng không kém phần hiệu quả mà còn giúp ta làm phong phú thêm vốn kiến thức cũng như kinh nghiệm giải quyết một bài toán nói chung.

Báo cáo này trình bày về bài toán xếp ba lô và một số hướng tiếp cận quen thuộc áp dụng cho việc giải bài toán. Do vốn kiến thức của bản thân còn hạn hẹp cũng như thời gian thực hiện có hạn, bài báo cáo không thể tránh khỏi những sai sót còn tồn tại, chúng em rất mong nhận được các ý kiến đóng góp cũng như nhận xét phản biện về nội dung để bài báo cáo được hoàn thiện và chính xác hơn.

Nhân đây, chúng em xin gửi lời cảm ơn sâu sắc đến PGS.TS. Nguyễn Thị Hồng Minh với những bài giảng về Thiết kế và đánh giá thuật toán cũng như những kinh nghiệm, kiến thức được cô trao đổi song song với những bài giảng trên lớp. Những điều ấy đã là nguồn động lực lớn thúc đẩy chúng em hoàn thành bài báo cáo này.

Ngày 1 tháng 6 năm 2021

Nhóm 05

Mục lục

Danh sách chương trình mẫu

Danh sách hình vẽ

Chương 1

Giới thiệu bài toán xếp ba lô

1.1 Bài toán xếp ba lô

Bài toán xếp ba lô(còn được biết đến với tên gọi bài toán cái túi) là một bài toán tối ưu hóa tổ hợp. Bài toán được đặt tên từ vấn đề chọn những gì quan trọng có thể nhét vừa vào trong một cái túi (với giới hạn khối lượng) để mang theo trong một chuyến đi. Các bài toán tương tự thường xuất hiện trong kinh doanh, toán tổ hợp, lý thuyết độ phức tạp tính toán, mật mã học và toán ứng dụng.

1.2 Nội dung bài toán

Trong siêu thị có N gói hàng, gói hàng thứ i có trọng lượng là W_i và giá trị là V_i. Một tên trộm đột nhập vào siêu thị, tên trộm mang theo một cái túi có thể mang được tối đa trọng lượng là M. Hỏi tên trộm sẽ lấy đi những gói hàng nào để được tổng giá trị là lớn nhất.

Dữ liệu nhập:

- Dòng thứ nhất là hai số N, M cách nhau một khoảng trắng ($1 \leq N \leq 100$, $1 \leq M \leq 100$)

- Trong N dòng tiếp theo, dòng thứ i là hai số nguyên W_i và V_i cách nhau một khoảng trắng ($1 \leq W_i$, $V_i \leq 100$)

Dữ liệu xuất:

- Nếu tên trộm không thể lấy được món đồ nào, in ra 0.

- Nếu tên trộm có thể lấy được ít nhất một món đồ, dòng thứ nhất in ra giá trị lớn nhất tên trộm có thể lấy. Dòng thứ hai là chỉ số những gói bị lấy. Nếu có nhiều cách lấy đồ có cùng giá trị lớn nhất, chỉ cần in ra một cách bất kỳ.

Dữ liệu mẫu:

Input:	Output:
5 11 3 3 4 4 5 4 9 10 4 5	12 5 2 1

Lấy 3 đồ vật 1, 2, 5 có tổng trọng là: $3+4+4=11$ và tổng giá trị là: $3+4+5=12$

Chương 2

Phương pháp Quay lui/Vét cạn

2.1 Hướng tiếp cận

Bài toán có thể được tóm tắt là đi tìm một cách chọn trong N gói hàng i gói có tổng trọng lượng không vượt quá M và tổng giá trị là lớn nhất. Dễ thấy đối với N gói ban đầu, trong phương án của ta đối với gói thứ i ta có thể chọn hoặc không chọn, suy nghĩ này cho ta một gợi ý về phương pháp giải bài toán.

Ta sẽ sinh các dãy nhị phân độ dài N để biểu diễn tất cả các tập con của N phần tử đại diện cho N gói hàng. Gọi dãy nhị phân thứ k là $S_k = a_1 \dots a_N$ với $a_i \in \{0, 1\}$, khi đó trong phương án chọn ta chọn gói hàng thứ i nếu $a_i = 1$ và ngược lại không chọn gói hàng thứ i nếu $a_i = 0$.

Sau khi xây dựng được tất cả các chuỗi nhị phân độ dài N ta tiến hành tính toán tổng giá trị và tổng trọng lượng cho từng phương án ứng với dãy nhị phân vừa được sinh.

Đáp án của bài toán sẽ là chuỗi nhị phân biểu diễn phương án thỏa mãn tổng trọng lượng không lớn hơn M và có tổng giá trị là lớn nhất.

Truy vết đáp án: Sau các bước so sánh và tìm ra được chuỗi nhị phân ứng với phương án tối ưu, ta chỉ việc dựa vào dãy nhị phân đó để suy ra phương án tối ưu sẽ gồm các gói hàng nào.

2.2 Tính đúng đắn

Do khi sinh tất cả các chuỗi nhị phân độ dài N, ta ngầm hiểu rằng sẽ đi tìm tất cả các khả năng chọn có thể xảy ra. Từ tập tất cả các phương án đó chọn ra phương án tối ưu nhất, phương án này đương nhiên là phương án tối ưu cho cả bài toán.

2.3 Phân tích tính hiệu quả

Quay lui/Vét cạn có thể là một trong các hướng tiếp cận thuộc lớp *Brute-force*, tức là ta tìm lời giải tối ưu của bài toán dựa vào cách tìm tất cả các phương án rồi so sánh chọn ra phương án tối ưu toàn cục.

Phương pháp này cho ta một cách tiếp cận dễ hiểu, dễ cài đặt. Nhược điểm của phương pháp là ta phải đi tìm tất cả các phương án có thể, ở đây là tất cả các dãy nhị phân độ dài N, con số này lên tới 2^N và dẫn tới việc bùng nổ tổ hợp khi N lớn.

Độ phức tạp về thời gian của hướng tiếp cận sẽ là $O(2^N)$ khi mà ta phải đi tính tất cả 2^N cấu hình. Ta sẽ duy trì một mảng để chứa cấu hình hiện tại, một mảng chứa cấu hình tối ưu hiện tại nên độ phức tạp về không gian lưu trữ sẽ là $O(N)$ cho 2 mảng có N phần tử.

Cải tiến: Ta có một nhận xét là có thể trong quá trình sinh cấu hình, cấu hình nào chưa sinh xong mà đã vi phạm về giới hạn trọng lượng hoặc không có khả năng tốt hơn cấu hình tối ưu hiện tại (theo nghĩa không thể đạt được tổng giá trị lớn hơn cấu hình tối ưu hiện tại) ta có thể bỏ qua và tiến hành xét tiếp cấu hình tiếp theo. Kỹ thuật này có tên là **Nhánh cận**(Branch and bound), sẽ làm giảm đáng kể độ phức tạp tính toán trong quá trình giải theo phương pháp Quay lui/ Vét cạn.

2.4 Cài đặt và kiểm thử

Mã mẫu sử dụng ngôn ngữ C++:

```cpp
#include<bits/stdc++.h>

using namespace std;

#define fast_io ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cout.tie(nullptr);
#define ll long long
#define inf 1000000000

int n, m, w[105], v[105], cur_state[105], opt_state[105];

int opt = 0, tmp_w = 0, tmp_v = 0;


void gen(int i) {
  if (i == n + 1) {
    if (tmp_w <= m && tmp_v > opt) {
      for (int k = 1; k <= n; ++k)opt_state[k] = cur_state[k];
      opt = tmp_v;
    }
  }
  else {
    tmp_w += w[i];
    tmp_v += v[i];
    cur_state[i] = 1;
    gen(i + 1);
    tmp_w -= w[i];
    tmp_v -= v[i];
    cur_state[i] = 0;
    gen(i + 1);
  }
}


int main() {

#ifndef ONLINE_JUDGE
  freopen("input.inp", "r", stdin);
  freopen("output.out", "w", stdout);
#endif
```

```
  fast_io

  cin >> n >> m;
  for (int i = 1; i <= n; ++i)cin >> w[i] >> v[i];

  gen(1);

  cout << opt << endl;

  for (int i = n; i > 0; --i) {
    if (opt_state[i])cout << i << " ";
  }

  return 0;
}
```

Chương trình 2.1: Mã C++ cài đặt theo phương pháp Quay lui/Vét cạn

Kết quả kiểm thử:

Các lần bạn nộp bài CATU2

ID	Thời gian nộp	Coder	Bài tập	Ngôn ngữ	Kết quả	Thời gian chạy	Bộ nhớ
913128	30/05/2021 07:59:39	ServantOfEvil	CATU2 - Cái túi 2	GNU C++11	Time limit exceed on test 12	1078 ms	1812 KB

Hình 2.1: Kết quả kiểm thử mã C++ theo phương pháp Quay lui/Vét cạn

```
#include<bits/stdc++.h>

using namespace std;

#define fast_io ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cout.tie(nullptr);
#define ll long long
#define inf 1000000000

int n, m, w[105], v[105], cur_state[105], opt_state[105];

int opt = 0, tmp_w = 0, tmp_v = 0;


void gen(int i) {
  if (i == n + 1) {
    if (tmp_w <= m && tmp_v > opt) {
      for (int k = 1; k <= n; ++k)opt_state[k] = cur_state[k];
      opt = tmp_v;
    }
  }
  else {
    tmp_w += w[i];
    tmp_v += v[i];
    cur_state[i] = 1;
    if (tmp_w <= m) gen(i + 1);
    tmp_w -= w[i];
    tmp_v -= v[i];
```

```
    cur_state[i] = 0;
    gen(i + 1);
  }
}


int main() {

#ifndef ONLINE_JUDGE
  freopen("input.inp", "r", stdin);
  freopen("output.out", "w", stdout);
#endif

  fast_io

  cin >> n >> m;
  for (int i = 1; i <= n; ++i)cin >> w[i] >> v[i];

  gen(1);

  cout << opt << endl;

  for (int i = n; i > 0; --i) {
    if (opt_state[i])cout << i << " ";
  }

  return 0;
}
```

Chương trình 2.2: Mã C++ cài đặt theo phương pháp Quay lui/Vét cạn có sử dụng Nhánh cận

Kết quả kiểm thử:

Các lần bạn nộp bài CATU2

ID	Thời gian nộp	Coder	Bài tập	Ngôn ngữ	Kết quả	Thời gian chạy	Bộ nhớ
913129	30/05/2021 08:01:12	ServantOfEvil	CATU2 - Cái túi 2	GNU C++11	Accepted	93 ms	1812 KB

Hình 2.2: Kết quả kiểm thử mã C++ theo phương pháp Quay lui/Vét cạn có áp dụng Nhánh cận

Chương 3

Phương pháp Đệ quy

3.1 Hướng tiếp cận

Dễ thấy ta có thể nhận được phương án tối ưu của bài toán bằng cách kết hợp phương án tối ưu của các bài toán con của nó. Cụ thể, tổng giá trị lớn nhất khi chọn trong i gói hàng sao cho tổng trọng lượng không vượt quá j bằng tổng giá trị lớn nhất đạt được khi chọn trong $i-1$ gói hàng sao cho tổng trọng lượng không vượt quá j nếu ta không lấy gói hàng thứ i hoặc tổng trọng lượng không vượt quá $j-W_i$ nếu ta lấy gói hàng thứ i cộng với giá trị tối ưu ở bước chọn thứ i.

Trường hợp suy biến hay trường hợp cơ sở của bài toán là tổng giá trị lớn nhất ta có thể đạt được khi chọn trong 0 gói hàng với giới hạn trọng lượng M hoặc chọn trong N gói hàng với giới hạn trọng lượng 0 đều bằng 0 $\forall M, N$.

Ta xây dựng được công thức truy hồi nghiệm của bài toán:

$$\begin{cases} f(0,j) = 0 \ \forall j = 1..M \\ f(i,0) = 0 \ \forall i = 1..N \\ f(i,j) = f(i-1,j) \ j < W_i \\ f(i,j) = max(f(i-1,j-W_i) + V_i, f(i-1,j)) \ j \geq W_i \end{cases} \tag{3.1}$$

Truy vết đáp án: Ta duy trì một bảng phương án chứa các kết quả của bài toán con $f(i,j)$:

- Nếu trong bước thứ i, ta chọn gói hàng thứ i thì $f(i,j) = f(i-1,j-W_i) + V_i$
- Nếu trong bước thứ i, ta không chọn gói hàng thứ i thì $f(i,j) = f(i-1,j)$

Vậy gói hàng thứ i được chọn trong phương án tối ưu khi $f(i,j) = f(i-1,j-W_i) + V_i$, ngược lại gói hàng thứ i không có trong phương án tối ưu - ta xét tiếp bước thứ $i-1$. Dựa vào nhận xét này ta có thể truy vết đáp án từ $f(N,M)$.

3.2 Tính đúng đắn

Vì ta đã xây dựng kết quả ở mỗi bước bằng cách lấy kết quả tối ưu ở các bước chọn trước rồi kết hợp với phương án tối ưu ở bước hiện tại nên hiển nhiên theo quy nạp ta có được kết quả tối ưu của cả bài toán ở bước chọn cuối cùng.

3.3 Phân tích tính hiệu quả

Với hướng tiếp cận sử dụng đệ quy, dựa vào công thức truy hồi ta thấy độ phức tạp thời gian của thuật toán là $O(2^N)$. Với yêu cầu lưu bảng truy vết ta mất $O(N.M)$ cho độ phức tạp về không gian.

Cải tiến: Ta có thể giảm độ phức tạp thời gian. Xét thấy thuật toán đệ quy cũ có thể sẽ đi tính lại nhiều lần kết quả của một bài toán con $f(i, j)$, điều này là không cần thiết mà chỉ làm tăng độ phức tạp tính toán. Ta có thể duy trì một bảng phương án để lưu các phương án tối ưu cục bộ đã được tính, với một bài toán con cần giải ta sẽ tra bảng phương án và chỉ đi giải khi trong bảng phương án chưa có lời giải cho bài toán con đó, trong phần này ta vừa sử dụng bảng phương án để truy vết đáp án, vừa dùng nó như là bảng ghi nhớ các phương án để tra cứu khi cần. Độ phức tạp thời gian được giảm xuống còn $O(N.M)$ ứng với trường hợp xấu nhất ta phải đi tính toàn bộ bảng phương án có N hàng và M cột.

3.4 Cài đặt và kiểm thử

Mã mẫu sử dụng ngôn ngữ C++:

```cpp
#include<bits/stdc++.h>

using namespace std;

#define fast_io ios_base::sync_with_stdio(false); cin.tie(nullptr); cout.tie(nullptr);
#define ll long long
#define inf 1000000000

int n, m, w[105], v[105];
int trace[105][105] = {0};

int solve(int i, int j) {
  if (j == 0 || i == 0)return 0;

  if (j >= w[i]) {
    int a = solve(i - 1, j);
    int b = solve(i - 1, j - w[i]) + v[i];
    trace[i][j] = max(a, b);
    return trace[i][j];
  }
  else {
    trace[i][j] = solve(i - 1, j);
    return trace[i][j];
  }
}

int main() {

#ifndef ONLINE_JUDGE
  freopen("input.inp", "r", stdin);
  freopen("output.out", "w", stdout);
#endif

  fast_io
```

```
35
36    cin >> n >> m;
37    for (int i = 1; i <= n; ++i)cin >> w[i] >> v[i];
38
39    cout << solve(n, m);
40
41    cout << endl;
42
43    while (trace[n][m] > 0) {
44
45      if (trace[n][m] == trace[n - 1][m - w[n]] + v[n]) {
46        cout << n << " ";
47        m -= w[n];
48      }
49      --n;
50    }
51
52      return 0;
53  }
```

Chương trình 3.1: Mã C++ cài đặt theo phương pháp Đệ quy

Kết quả kiểm thử:

Các lần bạn nộp bài CATU2

ID	Thời gian nộp	Coder	Bài tập	Ngôn ngữ	Kết quả	Thời gian chạy	Bộ nhớ
913130	30/05/2021 08:02:39	ServantOfEvil	CATU2 - Cái túi 2	GNU C++11	Accepted	203 ms	1852 KB

Hình 3.1: Kết quả kiểm thử mã C++ theo phương pháp Đệ quy

```
1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  #define fast_io ios_base::sync_with_stdio(false); cin.tie(nullptr);
       cout.tie(nullptr);
6  #define ll long long
7  #define inf 1000000000
8
9  int n, m, w[105], v[105];
10 int trace[105][105] = {0};
11
12 int solve(int i, int j) {
13   if (j == 0 || i == 0)return 0;
14
15   if (j >= w[i]) {
16     if (trace[i][j] == 0) {
17       int a = solve(i - 1, j);
18       int b = solve(i - 1, j - w[i]) + v[i];
19       trace[i][j] = max(a, b);
20     }
21     return trace[i][j];
22   }
23   else {
24
```

```cpp
    if (trace[i][j] == 0) {
      trace[i][j] = solve(i - 1, j);
    }
    return trace[i][j];
  }
}

int main() {

#ifndef ONLINE_JUDGE
  freopen("input.inp", "r", stdin);
  freopen("output.out", "w", stdout);
#endif

  fast_io

  cin >> n >> m;
  for (int i = 1; i <= n; ++i)cin >> w[i] >> v[i];

  cout << solve(n, m);

  cout << endl;

  while (trace[n][m] > 0) {

    if (trace[n][m] == trace[n - 1][m - w[n]] + v[n]) {
      cout << n << " ";
      m -= w[n];
    }
    --n;
  }

    return 0;
}
```

Chương trình 3.2: Mã C++ cài đặt theo phương pháp Đệ quy cải tiến

Kết quả kiểm thử:

Các lần bạn nộp bài CATU2

ID	Thời gian nộp	Coder	Bài tập	Ngôn ngữ	Kết quả	Thời gian chạy	Bộ nhớ
913131	30/05/2021 08:04:28	ServantOfEvil	CATU2 - Cái túi 2	GNU C++11	**Accepted**	15 ms	1888 KB

Hình 3.2: Kết quả kiểm thử mã C++ theo phương pháp Đệ quy cải tiến

Chương 4

Phương pháp Tham lam

4.1 Hướng tiếp cận

Có rất nhiều cách triển khai với hướng tiếp cận giải bài toán bằng Tham lam như tham lam theo trọng lượng, tham lam theo giá trị,... Ở đây ta sẽ xét một cách triển khai tư tưởng Tham lam hiệu quả hơn.

Nhận thấy có hai đại lượng chính cần tối ưu là trọng lượng của gói hàng và giá trị của nó. Ta có một ý tưởng đơn giản trong quá trình xây dựng phương án như sau: trong tất cả các gói hàng thì ta sẽ ưu tiên chọn gói hàng nào mà có trọng lượng nhỏ mà giá trị lớn trước hay nói cách khác là ưu tiên chọn gói hàng có tỉ lệ giá trị vượt trội so với tỉ lệ trọng lượng. Ta có thể đạt được ý đồ này bằng cách xét tỉ lệ V/W của các gói hàng với nhau.

Giữa hai gói hàng thứ i và thứ j, nếu ta có $V_i/W_i > V_j/W_j$, tức là gói hàng thứ i có giá trị vượt trội so với trọng lượng của nó hơn gói hàng thứ j, thì ta sẽ ưu tiên chọn gói hàng thứ i trước. Cứ như vậy ta sẽ chọn đến khi không thể chọn gói hàng nào nữa theo nghĩa tổng trọng lượng hiện tại không cho phép chọn thêm bất kì gói hàng nào hoặc các gói hàng đã được chọn hết.

Truy vết đáp án: Trong quá trình so sánh và chọn ta duy trì một danh sách lưu chỉ số của các gói hàng đã được chọn. Khi thuật toán kết thúc, danh sách đó sẽ chứa phương án tối ưu của bài toán.

4.2 Tính đúng đắn

Tham lam luôn là một hướng tiếp cận có độ ưu tiên cao khi bắt tay vào giải một bài toán tối ưu tổ hợp do tính đơn giản cũng như hiệu quả của nó. Tuy nhiên ta cần lưu ý là không phải lúc nào Tham lam cũng cho ta đáp án tối ưu chính xác nhất.

Với phương pháp áp dụng Tham lam cho bài toán ba lô là một ví dụ điển hình cho tính chất trên. Trong trường hợp này ta thường chỉ nhận được phương án xấp xỉ phương án tối ưu bởi không phải chiến lược chọn trên lúc nào cũng đạt được nghiệm tối ưu.

Ví dụ ta có đầu vào và đầu ra tương ứng khi áp dụng Tham lam như sau:

Input:	Output:
7 30	40
5 11	6 2 1
13 12	
9 8	
6 5	
14 8	
10 17	
12 6	

Trong ví dụ này phương pháp Tham lam cho ta phương án tối ưu với cách chọn các gói hàng thứ 6, 2, 1 có tổng trọng lượng là $10 + 13 + 5 = 28$ và nhận được tổng giá trị là $17 + 12 + 11 = 40$. Nhưng đây không phải phương án tối ưu của bài toán khi mà ta có thể chọn các gói hàng thứ 6, 4, 3, 1 với tổng trọng lượng là $10 + 6 + 9 + 5 = 30$ và tổng giá trị là $17 + 5 + 8 + 11 = 41$.

4.3 Phân tích tính hiệu quả

Mặc dù không phải lúc nào ta cũng nhận được đáp án tối ưu nhất khi sử dụng Tham lam nhưng trong thực tế rất ít khi ta phải tính chính xác phương án tối ưu nhất cho một bài toán, và vì vậy Tham lam luôn là sự lựa chọn tốt khi thể hiện ưu điểm vượt trội về tính đơn giản cũng như đòi hỏi thấp về chi phí thực hiện.

Với hướng tiếp cận sử dụng Tham lam như trên, phần tính toán chủ yếu của ta là sắp xếp không tăng dãy tỉ số V/W. Với bài toán sắp xếp thì hướng tiếp cận đơn giản nhất ta nghĩ đến là Bubble sort với độ phức tạp khi sắp xếp dãy N tỉ số là $O(N^2)$. Trong cài đặt ta sẽ dùng một thuật toán sắp xếp hiệu quả hơn là Quick sort với độ phức tạp $O(NlogN)$. Vậy bài toán được giải với độ phức tạp thời gian là $O(NlogN)$ với hướng tiếp cận tham lam. Ta cần lưu trữ một danh sách các gói hàng được chọn trong phương án tối ưu dùng để truy vết đáp án nên độ phức tạp về không gian lưu trữ sẽ là $O(N)$ cho tối đa N gói hàng.

4.4 Cài đặt và kiểm thử

Mã mẫu sử dụng ngôn ngữ C++:

```
#include<bits/stdc++.h>

using namespace std;

#define fast_io ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cout.tie(nullptr);
#define ll long long

int m, n;
struct vw
{
  float v, w;
  int idx;
};

vw a[105];

```

```cpp
set<int> res;

bool cmp(vw a, vw b) {
  return (a.v / a.w) > (b.v / b.w);
}

int main() {

#ifndef ONLINE_JUDGE
  freopen("input.inp", "r", stdin);
  freopen("output.out", "w", stdout);
#endif

  fast_io

  cin >> n >> m;

  for (int i = 0; i < n; ++i) {
    cin >> a[i].w >> a[i].v;
    a[i].idx = i + 1;
  }


  int ans = 0;

  sort(a, a + n, cmp);

  for (int i = 0; i < n; ++i) {
    if (m >= a[i].w) {
      m -= a[i].w;
      ans += a[i].v;
      res.insert(a[i].idx);
    }
  }


  cout << ans << endl;

  for (set<int>::reverse_iterator it = res.rbegin(); it != res.rend
   (); ++it) {
    cout << *it << " ";
  }

  return 0;

}
```

Chương trình 4.1: Mã C++ cài đặt theo phương pháp Tham lam

Chú thích:

- Ta định nghĩa một **Struct vw** lưu thông tin của một gói hàng gồm có trọng lượng w, giá trị v và số thứ tự idx.
- **res** dùng để lưu danh sách các gói hàng được chọn thông qua số thứ tự của chúng.
- Ta dùng **cmp** như một **Comparator** để tận dụng hàm sắp xếp có sẵn trong thư viện.

Kết quả kiểm thử:

Các lần bạn nộp bài CATU2

ID	Thời gian nộp	Coder	Bài tập	Ngôn ngữ	Kết quả	Thời gian chạy	Bộ nhớ
913132	30/05/2021 08:06:57	ServantOfEvil	CATU2 - Cái túi 2	GNU C++11	Wrong answer on test 5	15 ms	1056 KB

Hình 4.1: Kết quả kiểm thử mã C++ theo phương pháp Tham lam

Chương 5

Phương pháp Quy hoạch động

5.1 Hướng tiếp cận

Ta đi tìm công thức truy hồi của bài toán. Trước hết ta biểu diễn bài toán thành các trạng thái từ đó xây dựng được công thức truy hồi cho biết kết quả của một trạng thái dựa trên các trạng thái con của nó(Quy hoạch động trạng thái).

Ta gọi tổng giá trị lớn nhất tên trộm có thể đạt được bằng cách chọn trong i gói hàng với giới hạn trọng lượng không lớn hơn j là $f(i, j)$, giá trị món hàng thứ i là V_i, trọng lượng món hàng thứ i là W_i. Khi đó kết quả của bài toán sẽ là $f(N, M)$.

Với trạng thái bài toán bất kì chọn trong i món hàng với giới hạn trọng lượng j sao cho tổng giá trị lớn nhất thì ta có 2 trường hợp lựa chọn:

- Trong phương án tối ưu của ta, ta lấy món hàng thứ i nếu cái túi vẫn chứa được nó, tức là trọng lượng món hàng thứ i phải nhỏ hơn hoặc bằng j. Khi đó ta có:

$$f(i, j) = f(i - 1, j - W_i) + V_i \ (W_i \leq j) \tag{5.1}$$

- Trong phương án tối ưu ta không lấy món hàng thứ i, khi đó:

$$f(i, j) = f(i - 1, j) \tag{5.2}$$

Kết luận từ 2 trường hợp trên, ta muốn tìm tổng giá trị lớn nhất đạt được hay nói cách khác là đi tìm giá trị lớn nhất của hàm f. Ta thu được công thức truy hồi:

$$\begin{cases} f(i, j) = f(i - 1, j) \ (W_i > j) \\ f(i, j) = Max\Big[f(i - 1, j - W_i) + V_i,\ f(i - 1, j)\Big] \ (W_i \leq j) \end{cases} \tag{5.3}$$

Ta dễ dàng nhận thấy tổng giá trị lớn nhất có thể đạt được khi chọn trong 0 gói hàng với giới hạn túi chứa được trọng lượng j là 0. Vậy $f(0, j) = 0 \ \forall j \leq M$ là trường hợp suy biến của bài toán hay còn gọi là cơ sở quy hoạch động.

Truy vết đáp án: Với cách cài đặt $f(i, j)$ bằng bảng, ta có thể truy vết trên $f(i, j)$ giống như với hướng tiếp cận giải bài toán bằng Đệ quy.

5.2 Tính đúng đắn

Hướng tiếp cận theo phương pháp Quy hoạch động có điểm chung với phương pháp Đệ quy đã được chứng minh tính đúng đắn trong phần trước. Cả hai hướng tiếp cận đều chia nhỏ và kết hợp kết quả từ các bài toán con đến khi bài toán con là bài toán con cơ sở và dễ dàng thu được nghiệm tối ưu. Quy hoạch động có một ưu điểm vượt trội so với Đệ quy khi với một bài toán con đã được giải trước đó ta có luôn kết quả mà không cần phải đi giải lại bài toán con đó. Chính điều này tạo nên sự hiệu quả trong quá trình tính toán với hướng tiếp cận sử dụng Quy hoạch động

Một điểm cộng so với Đệ quy là với phương pháp Quy hoạch động ta sẽ xác định trên bảng phương án một thứ tự tính toán, từ đó ta có thể tính toán một cách hiệu quả bảng phương án mà không cần sử dụng đệ quy. Việc này có ý nghĩa khá quan trọng khi ta biết rằng thủ tục gọi đệ quy có chi phí về bộ nhớ rất đắt đỏ và không thể thực hiện đệ quy quá sâu.

5.3 Phân tích tính hiệu quả

So với các thuật toán đã nêu, Quy hoạch động dường như là phương pháp hiệu quả nhất cả về thời gian và không gian lưu trữ. Độ phức tạp thời gian của thuật toán theo hướng tiếp cận Quy hoạch động là $O(N.M)$ với bài toán trọng lượng và giá trị của các món hàng là số nguyên. Độ phức tạp không gian cũng là $O(N.M)$ khi ta phải duy trì một bảng phương án ứng với hàm f của bài toán với N món hàng và giới hạn trọng lượng của túi là M.

Hướng tiếp cận này tỏ ra không phù hợp đối với trường hợp trọng lượng các gói hàng là số thực hoặc giới hạn về dữ liệu đầu vào lớn. Khi đó ta không thể lưu trữ và tính toán bảng phương án một cách hiệu quả.

5.4 Cài đặt và kiểm thử

Mã mẫu sử dụng ngôn ngữ C++:

```
#include<bits/stdc++.h>

using namespace std;

#define fast_io ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cout.tie(nullptr);
#define ll long long

int n, m, w[105], v[105], f[105][105];

int main() {

#ifndef ONLINE_JUDGE
  freopen("input.inp", "r", stdin);
  freopen("output.out", "w", stdout);
#endif

  fast_io

  cin >> n >> m;
```

```cpp
  for (int i = 1; i <= n; ++i)cin >> w[i] >> v[i];

  memset(f, 0, sizeof(f));

  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      f[i][j] = f[i - 1][j];
      if (j >= w[i])f[i][j] = max(f[i][j], f[i - 1][j - w[i]] + v[i
   ]);
    }
  }

  cout << f[n][m] << "\n";

  while (f[n][m] > 0) {

   if (f[n][m] == f[n - 1][m - w[n]] + v[n]) {
      cout << n << " ";
      m -= w[n];
    }
    --n;
  }
    return 0;
}
```

Chương trình 5.1: Mã C++ cài đặt theo phương pháp Quy hoạch động

Chú thích:

- n: số lượng gói hàng, m: trọng lượng giới hạn
- $w[i]$: trọng lượng gói hàng thứ i
- $v[i]$: giá trị gói hàng thứ i
- $f[i][j]$: tổng giá trị lớn nhất có thể đạt được khi chọn trong i gói và giới hạn trọng lượng j

Kết quả kiểm thử:

Các lần bạn nộp bài CATU2

ID	Thời gian nộp	Coder	Bài tập	Ngôn ngữ	Kết quả	Thời gian chạy	Bộ nhớ
913133	30/05/2021 08:08:09	ServantOfEvil	CATU2 - Cái túi 2	GNU C++11	Accepted	15 ms	1896 KB

Hình 5.1: Kết quả kiểm thử mã C++ theo phương pháp Quy hoạch động

Chương 6

Tổng kết và nhận xét

Bài toán xếp ba lô là một bài toán tối ưu điển hình và được coi như bài học vỡ lòng về phương pháp Quy hoạch động. Tuy nhiên, ngoài Quy hoạch động ta còn có rất nhiều phương pháp khác nhau để giải quyết bài toán.

Hướng tiếp cận sử dụng Quay lui/Vét cạn có lẽ là sự lựa chọn đơn giản, thô sơ và dễ tiếp cận nhất. Đổi lại, đối với phương pháp này độ phức tạp thời gian là $O(2^N)$, tỏ ra không hiệu quả với kích thước dữ liệu đầu vào lớn. Quay lui/Vét cạn dựa trên tư tưởng *Brute-force* tức là xét hết tất cả các trường hợp có thể xảy ra của bài toán rồi chọn từ tập cấu hình đó cấu hình tối ưu nhất.

Phương pháp Đệ quy cho ta một cái nhìn mới về bài toán, là một hướng tiếp cận hay và hiệu quả. Cách ta định nghĩa quan hệ truy hồi giữa các bài toán con rồi xây dựng nghiệm tối ưu cho bài toán gốc cho thấy sự tinh tế trong việc giải quyết bài toán. Với cách cài đặt gốc thì Đệ quy cũng có độ phức tạp thời gian là $O(2^N)$ và, cũng như phương pháp Quay lui/Vét cạn, tỏ ra kém hiệu quả với đầu vào dữ liệu lớn. Hơn nữa, thao tác gọi đệ quy được cho là có chi phí bộ nhớ rất đắt đỏ, ta không thể phụ thuộc quá nặng vào cấu trúc đệ quy khi mà tài nguyên bộ nhớ có hạn. Ta cũng có một cải tiến đáng kể độ phức tạp thời gian cho phương pháp Đệ quy bằng cách lưu trữ bảng phương án và thực hiện tra bảng phương án trước khi tính toán một bài toán con nào đó, độ phức tạp được giảm xuống còn $O(N.M)$.

Khác với hai phương pháp trên, Tham lam là một cách tiếp cận có thể nói là gần gũi với tư duy con người nhất. Hầu hết trong thực tế khi ta gặp phải một vấn đề tối ưu nào đó, Tham lam sẽ là suy nghĩ đầu tiên xuất hiện. Tham lam là một phương pháp có hiệu quả tính toán vượt trội so với các hướng tiếp cận còn lại. Tuy nhiên ta cần lưu ý về độ chính xác của Tham lam, Tham lam không phải lúc nào cũng cho ta kết quả tối ưu nhất. Bởi thế Tham lam là hướng tiếp cận tốt khi ta chỉ cần xấp xỉ phương án tối ưu và vì lí do này Tham lam được áp dụng rộng rãi trong thực tế. Với cách cài đặt ý tưởng Tham lam đã nêu, thủ tục sắp xếp sử dụng Quick sort có giá là $O(NlogN)$, độ phức tạp thời gian của thủ tục duyệt chọn là $O(N)$, vậy ta có độ phức tạp của cả chương trình theo phương pháp Tham lam là $O(NlogN)$ - một độ phức tạp tính toán lí tưởng.

Như đã nói, Quy hoạch động thường là phương pháp được nghĩ đến đầu tiên khi nhắc đến bài toán xếp ba lô. Bài toán xếp ba lô là bài toán điển hình cho ví dụ về phương pháp Quy hoạch động. Quy hoạch động dựa trên công thức truy hồi quan hệ nghiệm của các bài toán con với bài toán gốc, giống với Đệ quy. Điều đặc biệt ở đây là, trong Quy hoạch động ta sẽ xác định một thứ tự tính toán và do đó ta có thể loại bỏ được gánh nặng tính toán của thủ tục đệ quy, đặc điểm này khiến

Quy hoạch động trở thành một thuật toán mạnh mẽ giải quyết bài toán tối ưu. Với Quy hoạch động, độ phức tạp thời gian cần thiết là $O(N.M)$, là một độ phức tạp lí tưởng ứng với nghiệm tối ưu chính xác mà ta cần.

Các kết quả được cô đọng lại trong bảng sau:

Phương pháp	Ưu điểm	Nhược điểm	Độ p.t thời gian	Độ p.t không gian
Quay lui/Vét Cạn	+ Đơn giản + Dễ tiếp cận + Cho kết quả chính xác	+ Độ phức tạp lớn + Không hiệu quả với đầu vào lớn	$O(2^N)$	$O(N)$
Đệ quy	+ Hiệu quả + Cho kết quả chính xác	+ Độ phức tạp lớn với cách cài đặt thô sơ + Phụ thuộc vào thủ tục gọi đệ quy với chi phí đắt đỏ	+ $O(2^N)$ với cách cài đặt thô sơ + $O(N.M)$ với cách cài đặt cải tiến	$O(N.M)$
Tham lam	+ Độ phức tạp tính toán nhỏ + Hiệu quả khi chỉ cần xấp xỉ phương án tối ưu + Gần gũi với tư duy con người	Không cho kết quả chính xác trong đa số trường hợp	Phụ thuộc vào độ phức tạp thủ tục sắp xếp: + $O(N^2)$ với cách cài đặt dùng Bubble sort + $O(NlogN)$ với cách cài đặt dùng Quick sort	$O(N)$
Quy hoạch động	+ Hiệu quả + Cho kết quả nhanh, chính xác	+ Cần cách lưu trữ trạng thái bài toán hiệu quả + Cần xác định được thứ tự tính toán bằng phương án	$O(N.M)$	$O(N.M)$

Bảng 6.1: So sánh giữa các phương pháp giải

Ngoài các phương pháp đã trình bày, với bài toán xếp ba lo ta có rất nhiều hướng giải, mỗi hướng tiếp cận đều có những ưu nhược điểm và cái hay riêng. Bài báo cáo này trình bày những phương pháp quen thuộc và cơ bản nhất, mang lại cho người đọc cái nhìn tổng quát về "nghệ thuật" giải bài toán tối ưu tổ hợp.

Tài liệu tham khảo

[1] Nguyễn Thị Hồng Minh. "Thiết kế và đánh giá thuật toán" - *Bài giảng học phần MAT3504*. ĐH Khoa học Tự nhiên Hà Nội(2021).
[2] http://ntucoder.net/Problem/Details/1148 - *Bài toán cái túi, đề bài và test cases.*
[3] https://en.wikipedia.org/wiki/Knapsack_problem - *Bài toán xếp ba lô.*