# Số cách chia N phần tử khác nhau vào K nhóm

##### **Đề bài**

Cho N phần tử khác nhau, hỏi có bao nhiêu cách chia N phần tử đó thành K nhóm, mà mỗi nhóm có ít nhất 1 phần tử (các hoán vị của nhóm được xem là 1 cách).

**Giới hạn:**

* 1 ≤ K ≤ N ≤ 25

##### **Định dạng test**

**Input:**

* Dòng đầu tiên chứa số T là số test.
* T dòng tiếp theo mỗi dòng chứa 2 số N và K.

**Output:**

* T dòng, mỗi dòng là số cách với test tương ứng.

Ví dụ:

input:

1

4 2

---

output:

7

Giải thích : 7 cách chia đó là (ABC)(D) , (ABD)(C) , (ADC)(B) , (DBC)(A) , (AB)(CD) , (AC)(BD) , (BC)(AD).

##### **Thuật toán**

Gọi f[i][j] là số cách chia i phần tử vào j nhóm, công thức truy hồi:

f[i][j] = f[i-1][j-1] + f[i-1][j] \* j

Giải thích:

* Nếu đã chia được i-1 phần tử trước đó thành j-1 nhóm, thì phần tử thứ i sẽ chỉ có thể được chia vào nhóm thứ j.
* Nếu đã chia được i-1 phần tử trước đó thành j nhóm, thì sẽ có j cách để chia phần tử thứ i vào j nhóm đó.

**Phân tích**

**1. Chia i phần tử thành j nhóm:**

Giả sử chúng ta đã chia i-1 phần tử thành j nhóm, và bây giờ chúng ta thêm phần tử thứ i vào nhóm.

Có hai cách để thêm phần tử thứ i vào các nhóm hiện có:

**Cách 1: Tạo một nhóm mới với phần tử i:**

* Nếu chúng ta thêm phần tử i vào một nhóm mới, thì chúng ta phải chia i-1 phần tử thành j-1 nhóm.
* Số cách thực hiện điều này là f[i-1][j-1].

**Cách 2: Thêm phần tử i vào một trong các nhóm hiện có:**

* Nếu chúng ta thêm phần tử i vào một trong j nhóm hiện có, thì chúng ta cần chia i-1 phần tử thành j nhóm và phần tử i sẽ được thêm vào một trong j nhóm đó.
* Số cách thực hiện điều này là f[i-1][j] \* j.

**Tổng hợp lại:**

Số cách chia i phần tử thành j nhóm bằng tổng số cách của hai cách trên:

f[i][j]=f[i−1][j−1]+f[i−1][j]×jf[i][j] = f[i-1][j-1] + f[i-1][j] \times jf[i][j]=f[i−1][j−1]+f[i−1][j]×j

* **f[i-1][j-1]**: Số cách chia i-1 phần tử thành j-1 nhóm và thêm phần tử i vào nhóm mới.
* **f[i-1][j] \* j**: Số cách chia i-1 phần tử thành j nhóm và thêm phần tử i vào một trong các nhóm hiện có.

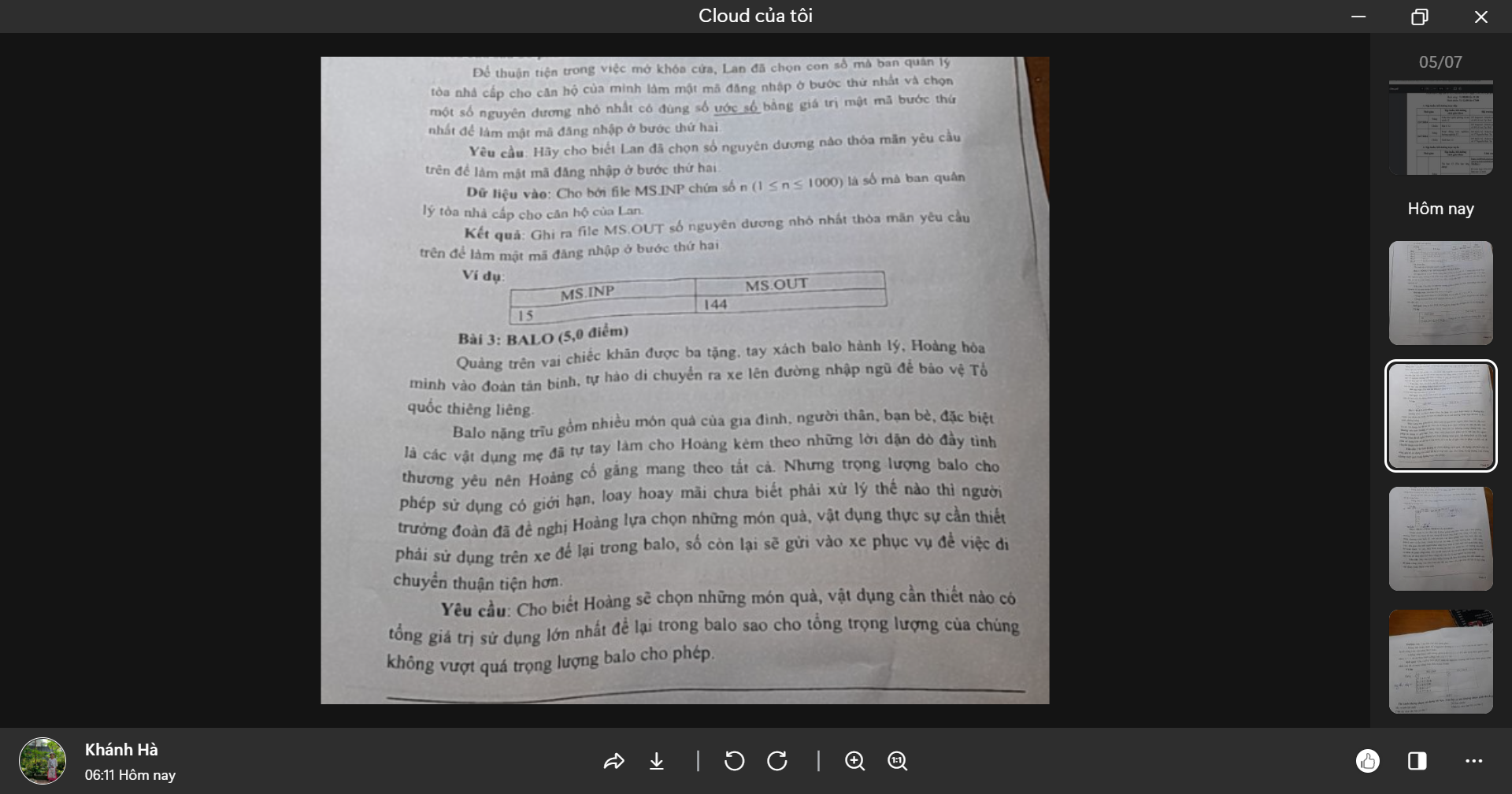
**Ví dụ cụ thể**

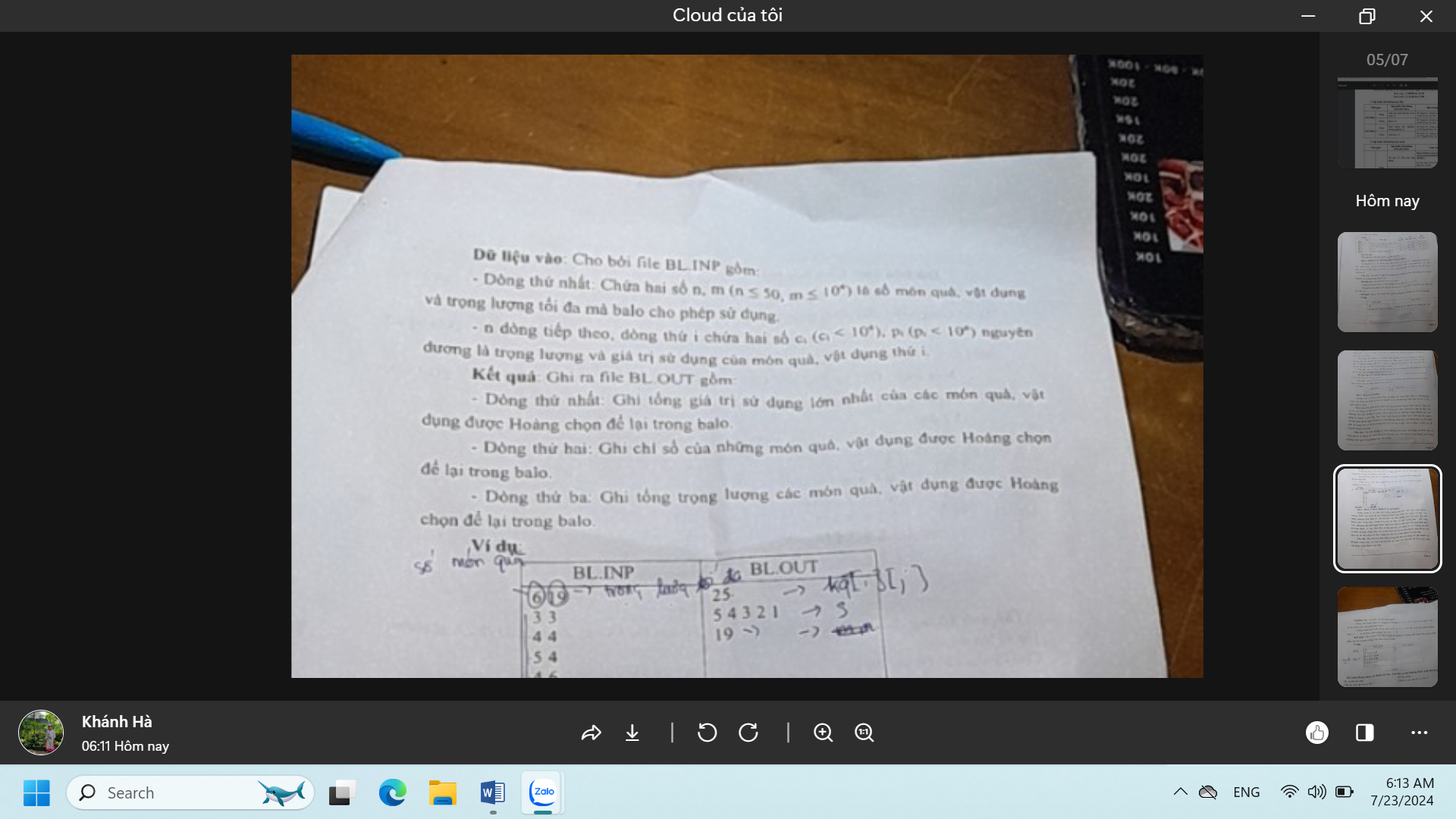
Hãy xem xét ví dụ với 4 phần tử và 2 nhóm (N = 4, K = 2):

**Trường hợp i = 4 và j = 2:**

* **Trường hợp 1:** Chia 3 phần tử thành 1 nhóm và thêm phần tử thứ 4 vào nhóm mới (tạo nhóm thứ hai). Số cách là f[3][1].
* **Trường hợp 2:** Chia 3 phần tử thành 2 nhóm và thêm phần tử thứ 4 vào một trong hai nhóm hiện có. Số cách là f[3][2] \* 2.

BÀI TOÁN CÁI TÚI



Bl.Inp bl.out

6 19 25

3 3 5 4 3 2 1

4 4 19

5 4

4 6

3 8

9 1

**Công thức quy hoạch động**

Chúng ta sử dụng một mảng hai chiều dp để lưu trữ kết quả của các bài toán con. Ở đây, dp[i][j] biểu diễn giá trị lớn nhất có thể đạt được với i vật phẩm và trọng lượng không vượt quá j.

**Cách thiết lập công thức**

1. **Khởi tạo mảng dp:** Mảng dp được khởi tạo với kích thước (n+1) x (m+1), với n là số lượng vật phẩm và m là trọng lượng tối đa của balo. Ban đầu, tất cả các phần tử của dp đều được gán giá trị 0.

int dp[n+1][m+1];

memset(dp, 0, sizeof(dp));

1. **Điền bảng dp:** Sử dụng hai vòng lặp lồng nhau để điền giá trị vào bảng dp. Vòng lặp bên ngoài chạy qua từng vật phẩm (i từ 1 đến n), và vòng lặp bên trong chạy qua từng khả năng trọng lượng (j từ 1 đến m).

for (i = 1; i <= n; i++) {

for (j = 1; j <= m; j++) {

dp[i][j] = dp[i-1][j]; // Không chọn vật phẩm thứ i

if (j >= w[i] && dp[i][j] < dp[i-1][j-w[i]] + v[i]) {

dp[i][j] = dp[i-1][j-w[i]] + v[i]; // Chọn vật phẩm thứ i

}

}

}

* + dp[i][j] = dp[i-1][j]: Nếu không chọn vật phẩm thứ i, giá trị lớn nhất với i vật phẩm và trọng lượng j sẽ là giá trị lớn nhất với i-1 vật phẩm và trọng lượng j.
  + if (j >= w[i] && dp[i][j] < dp[i-1][j-w[i]] + v[i]): Nếu có thể chọn vật phẩm thứ i (trọng lượng j đủ lớn để chứa vật phẩm i), và giá trị mới lớn hơn giá trị cũ, thì cập nhật giá trị mới cho dp[i][j].

1. **Xuất giá trị lớn nhất:** Giá trị lớn nhất có thể đạt được với n vật phẩm và trọng lượng tối đa m là dp[n][m].

cout << dp[n][m] << endl;

1. **Truy vết để tìm ra các vật phẩm đã được chọn:** Để xác định những vật phẩm nào đã được chọn, ta truy ngược lại từ dp[n][m]. Nếu dp[n][m] != dp[n-1][m], điều đó có nghĩa là vật phẩm n đã được chọn.

int s = 0;

while (n != 0) {

if (dp[n][m] != dp[n-1][m]) {

cout << n << " ";

s += w[n];

m -= w[n];

}

n--;

}

cout << endl;

cout << s;

* + Nếu dp[n][m] != dp[n-1][m], nghĩa là vật phẩm n đã được chọn, ta in ra n, cộng trọng lượng của vật phẩm này vào tổng trọng lượng đã chọn (s), và giảm m theo trọng lượng của vật phẩm này.
  + Tiếp tục giảm n và lặp lại cho đến khi n bằng 0.

CODE MẪU

#include<bits/stdc++.h>

#define nmax 10000

using namespace std;

int w[nmax], v[nmax], n, m;

// Hàm giải quyết bài toán knapsack

void xuli() {

int i, j;

int dp[n+1][m+1];

memset(dp, 0, sizeof(dp));

// Điền bảng dp

for (i = 1; i <= n; i++) {

for (j = 1; j <= m; j++) {

dp[i][j] = dp[i-1][j];

if (j >= w[i] )

{

dp[i][j] =max(dp[i][j], dp[i-1][j-w[i]] + v[i]);

}

}

}

// Xuất giá trị lớn nhất

cout << dp[n][m] << endl;

// Truy vết để tìm ra các vật phẩm đã được chọn

int s = 0;

while (n != 0) {

if (dp[n][m] != dp[n-1][m])

{

cout << n << " ";

s += w[n];

m -= w[n];

}

n--;

}

cout << endl;

cout << s;

}

int main() {

int i;

cin >> n >> m;

for (i = 1; i <= n; i++) {

cin >> w[i] >> v[i];

}

xuli();

return 0;

}

**Giải thích**

1. **Nhập Dữ Liệu**: Chương trình đọc n (số lượng vật phẩm) và m (trọng lượng tối đa của balo). Sau đó đọc trọng lượng (w[i]) và giá trị (v[i]) của từng vật phẩm.
2. **Bảng Động (Dynamic Programming Table)**: dp[i][j] lưu trữ giá trị lớn nhất có thể đạt được với i vật phẩm đầu tiên và sức chứa của balo là j.
3. **Khởi Tạo Bảng**: Bảng được khởi tạo bằng 0.
4. **Điền Bảng**: Vòng lặp lồng nhau điền vào bảng dựa trên việc vật phẩm hiện tại i có thể vừa trong balo (j >= w[i]) và liệu việc thêm nó có làm tăng giá trị hay không.
5. **Xuất Kết Quả**: Giá trị lớn nhất có thể đạt được sẽ được in ra.
6. **Truy Vết**: Mã truy vết lại để tìm ra những vật phẩm nào được chọn trong giải pháp tối ưu và in chúng cùng với tổng trọng lượng.

### Dữ Liệu Đầu Vào (Bl.Inp)

6 19

3 3

4 4

5 4

4 6

3 8

9 1

### Chạy Tay

#### Bước 1: Khởi tạo bảng dp và các biến cần thiết

n = 6 (số lượng vật phẩm)

m = 19 (trọng lượng tối đa của balo)

w = {0, 3, 4, 5, 4, 3, 9} (trọng lượng các vật phẩm, bắt đầu từ 1 để dễ truy cập)

v = {0, 3, 4, 4, 6, 8, 1} (giá trị các vật phẩm, bắt đầu từ 1 để dễ truy cập)

dp[n+1][m+1] = dp[7][20]

#### Bước 2: Khởi tạo bảng dp bằng 0

dp[0][j] = 0 cho mọi j

dp[i][0] = 0 cho mọi i

#### Bước 3: Điền bảng dp

Điền bảng từ i=1 đến i=6, và j=1 đến j=19:

1. **i=1**:
   * j từ 1 đến 2: dp[1][j] = 0 (trọng lượng 3 không vừa)
   * j từ 3 đến 19: dp[1][j] = max(dp[0][j], dp[0][j-3] + 3) = 3
2. **i=2**:
   * j từ 1 đến 3: dp[2][j] = dp[1][j]
   * j từ 4 đến 6: dp[2][j] = max(dp[1][j], dp[1][j-4] + 4)
   * j từ 7 đến 19: dp[2][j] = max(dp[1][j], dp[1][j-4] + 4) = 7
3. **i=3**:
   * j từ 1 đến 4: dp[3][j] = dp[2][j]
   * j từ 5 đến 9: dp[3][j] = max(dp[2][j], dp[2][j-5] + 4)
   * j từ 10 đến 19: dp[3][j] = max(dp[2][j], dp[2][j-5] + 4) = 11
4. **i=4**:
   * j từ 1 đến 4: dp[4][j] = dp[3][j]
   * j từ 5 đến 7: dp[4][j] = max(dp[3][j], dp[3][j-4] + 6)
   * j từ 8 đến 19: dp[4][j] = max(dp[3][j], dp[3][j-4] + 6) = 17
5. **i=5**:
   * j từ 1 đến 2: dp[5][j] = dp[4][j]
   * j từ 3 đến 7: dp[5][j] = max(dp[4][j], dp[4][j-3] + 8)
   * j từ 8 đến 19: dp[5][j] = max(dp[4][j], dp[4][j-3] + 8) = 25
6. **i=6**:
   * j từ 1 đến 8: dp[6][j] = dp[5][j]
   * j từ 9 đến 19: dp[6][j] = max(dp[5][j], dp[5][j-9] + 1) = 25

Kết quả bảng dp:

css

Copy code

dp[6][19] = 25

#### Bước 4: Truy vết để tìm ra các vật phẩm đã được chọn

Bắt đầu từ dp[6][19]:

1. dp[6][19] = dp[5][19] nên không chọn vật phẩm 6.
2. dp[5][19] = dp[4][16] + 8 nên chọn vật phẩm 5, m = 19 - 3 = 16.
3. dp[4][16] = dp[3][12] + 6 nên chọn vật phẩm 4, m = 16 - 4 = 12.
4. dp[3][12] = dp[2][7] + 4 nên chọn vật phẩm 3, m = 12 - 5 = 7.
5. dp[2][7] = dp[1][3] + 4 nên chọn vật phẩm 2, m = 7 - 4 = 3.
6. dp[1][3] = dp[0][0] + 3 nên chọn vật phẩm 1, m = 3 - 3 = 0.

Kết quả là các vật phẩm được chọn: 1, 2, 3, 4, 5 với tổng giá trị là 25 và tổng trọng lượng là 3 + 4 + 5 + 4 + 3 = 19.

### Kết Quả Đầu Ra (bl.out)

25

5 4 3 2 1

19

Như vậy, bộ dữ liệu đầu vào cho kết quả đầu ra là giá trị lớn nhất 25 và các vật phẩm được chọn là 5, 4, 3, 2, 1 với tổng trọng lượng là 19.

Dưới đây là bảng phương án động (dp) cho bài toán balo với dữ liệu đầu vào đã cung cấp. Mỗi ô dp[i][j] đại diện cho giá trị lớn nhất có thể đạt được khi sử dụng các vật phẩm từ 1 đến i với trọng lượng tối đa là j.

**Bảng Phương Án (dp)**

Mỗi hàng trong bảng tương ứng với một vật phẩm, mỗi cột tương ứng với một trọng lượng từ 0 đến 19.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

-------------------------------------------------------------

0 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

1 | 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

2 | 0 0 0 3 4 4 4 7 7 7 7 7 7 7 7 7 7 7 7 7

3 | 0 0 0 3 4 4 4 7 7 7 8 8 8 11 11 11 11 11 11 11

4 | 0 0 0 3 4 4 6 7 9 10 10 10 13 13 13 15 16 17 17 17

5 | 0 0 0 3 4 4 6 7 11 12 12 14 15 15 18 19 19 21 22 22

6 | 0 0 0 3 4 4 6 7 11 12 12 14 15 15 18 19 19 21 22 25

**Giải Thích Bảng Phương Án**

* Hàng đầu tiên (i=0) đại diện cho trường hợp không có vật phẩm nào, do đó mọi giá trị đều bằng 0.
* Cột đầu tiên (j=0) đại diện cho trường hợp trọng lượng tối đa là 0, do đó mọi giá trị đều bằng 0.
* Các giá trị trong các ô dp[i][j] đại diện cho giá trị lớn nhất có thể đạt được khi chọn từ 1 đến i vật phẩm và trọng lượng tối đa là j.

**Truy Vết Lại Các Vật Phẩm Được Chọn**

Bắt đầu từ dp[6][19], ta sẽ truy vết ngược lại để tìm các vật phẩm đã chọn:

1. dp[6][19] = dp[5][19] + 1 => không chọn vật phẩm 6 (giá trị không thay đổi).
2. dp[5][19] = dp[4][16] + 8 => chọn vật phẩm 5 (trọng lượng giảm từ 19 xuống 16).
3. dp[4][16] = dp[3][12] + 6 => chọn vật phẩm 4 (trọng lượng giảm từ 16 xuống 12).
4. dp[3][12] = dp[2][7] + 4 => chọn vật phẩm 3 (trọng lượng giảm từ 12 xuống 7).
5. dp[2][7] = dp[1][3] + 4 => chọn vật phẩm 2 (trọng lượng giảm từ 7 xuống 3).
6. dp[1][3] = dp[0][0] + 3 => chọn vật phẩm 1 (trọng lượng giảm từ 3 xuống 0).

**Kết Quả Cuối Cùng**

* Các vật phẩm được chọn: 1, 2, 3, 4, 5
* Tổng giá trị: 25
* Tổng trọng lượng: 19
* BÀI TOÁN CHIA NHÓM

#include <bits/stdc++.h>

using namespace std;

int n, k, t;

long long f[100][100];

int main()

{

f[0][0] = 1;

for (int i=1;i<=25;i++)

for (int j=1;j<=i;j++)

f[i][j] = f[i-1][j-1] + f[i-1][j]\*j;

cin>>t;

while (t--)

{

cin>>n>>k;

cout<<f[n][k];

}

return 0;

}

*Có N gói kẹo, gói thứ i có A*i *cái kẹo. Không được bóc bất kỳ một gói kẹo nào, cần chia N gói kẹo thành hai phần sao cho độ chênh lệch số kẹo giữa hai gói là ít nhất.*

*Dữ liệu vào trong file “****chiakeo.inp****” có dạng*:

*– Dòng đầu tiên là số N(N<=100);*

*– Dòng thứ hai là N số A*i*(i=1, 2,.., N; A*i*<=100).*

*Kết quả ra file “****chiakeo.out****” có dạng*:

*– Dòng đầu là độ chênh lệch nhỏ nhất giữa hai phần có thể được.*

*– 2 dòng tiếp theo là 2 đoạn đã được chia ra sao cho chênh lệch là ít nhất*

Chiakeo.Inp chiakeo.out

7 1

7 9 14 18 15 3 7 9 18 3 7

7 14 15

Công thức này là một phần của thuật toán quy hoạch động để giải bài toán phân chia các gói kẹo sao cho độ chênh lệch nhỏ nhất giữa hai phần. Để hiểu rõ hơn, ta sẽ phân tích từng bước của thuật toán.

### Ý tưởng chính của thuật toán

Bài toán có thể được coi là bài toán con của bài toán "Subset Sum Problem", nơi mà ta muốn kiểm tra xem liệu có thể chia tập hợp các phần tử thành hai phần có tổng gần nhau nhất.

Chúng ta sử dụng mảng f để biểu diễn trạng thái của bài toán. f[j] sẽ lưu trữ chỉ số của gói kẹo cuối cùng được sử dụng để đạt được tổng j. Nếu f[j] == -1, điều đó có nghĩa là không thể đạt được tổng j với các gói kẹo hiện tại.

### Bước 1: Khởi tạo

cpp

Copy code

f[0] = 0;

Điều này có nghĩa là chúng ta có thể đạt được tổng 0 mà không cần sử dụng bất kỳ gói kẹo nào.

### Bước 2: Cập nhật mảng f

cpp

Copy code

for (int i = 1; i <= n; i++) {

for (int j = t; j >= a[i]; j--) {

if (f[j - a[i]] != -1 && f[j] == -1) {

f[j] = i;

}

}

}

Ở đây, ta lặp qua từng gói kẹo a[i] và cố gắng cập nhật mảng f để bao gồm gói kẹo a[i]. Việc cập nhật từ phải sang trái (for (int j = t; j >= a[i]; j--)) đảm bảo rằng mỗi gói kẹo chỉ được sử dụng một lần.

#### Chi tiết từng bước:

* for (int i = 1; i <= n; i++): Lặp qua từng gói kẹo.
* for (int j = t; j >= a[i]; j--): Lặp qua từng tổng có thể từ t đến a[i].
* if (f[j - a[i]] != -1 && f[j] == -1): Kiểm tra nếu có thể đạt được tổng j - a[i] với các gói kẹo hiện tại (f[j - a[i]] != -1) và tổng j chưa được đạt được (f[j] == -1).
* f[j] = i;: Nếu điều kiện trên thỏa mãn, cập nhật f[j] bằng chỉ số của gói kẹo a[i].

### Tại sao công thức này hoạt động

Công thức này hoạt động dựa trên nguyên lý quy hoạch động, nơi mà chúng ta xây dựng nghiệm của bài toán lớn từ nghiệm của các bài toán con nhỏ hơn.

1. **Khởi tạo**: f[0] = 0 vì chúng ta luôn có thể đạt được tổng 0 mà không cần sử dụng bất kỳ gói kẹo nào.
2. **Cập nhật**: Khi duyệt qua từng gói kẹo, chúng ta kiểm tra tất cả các tổng có thể đạt được từ t đến a[i]. Nếu có thể đạt được tổng j - a[i], thì có thể đạt được tổng j bằng cách thêm gói kẹo a[i] vào. Chúng ta chỉ cập nhật f[j] nếu f[j] chưa được cập nhật trước đó để đảm bảo rằng mỗi gói kẹo chỉ được sử dụng một lần.
3. **Truy vết**: Sau khi cập nhật mảng f, chúng ta có thể truy vết lại để tìm các gói kẹo đã được sử dụng để đạt được tổng gần nhất với t.
4. **Đọc dữ liệu từ tệp**:

cpp

Copy code

freopen("chiakeo.inp", "r", stdin);

freopen("chiakeo.out", "w", stdout);

Đoạn mã này sử dụng freopen để thay thế cin và cout bằng các tệp "chiakeo.inp" và "chiakeo.out" để đọc dữ liệu đầu vào và ghi dữ liệu đầu ra.

1. **Khởi tạo và đọc dữ liệu**:

cpp

Copy code

int n, t = 0;

cin >> n;

int a[n + 1];

for (int i = 1; i <= n; i++) {

cin >> a[i];

t += a[i];

}

Đọc số lượng phần tử n và mảng a từ tệp. Tính tổng tất cả các phần tử t.

1. **Tính toán giá trị t/2**:

cpp

Copy code

t = t / 2;

Đặt t là tổng của một nửa tổng các phần tử, vì mục tiêu là chia tập hợp thành hai nhóm với tổng gần bằng nhau.

1. **Khởi tạo mảng DP và các biến phụ trợ**:

cpp

Copy code

int f[t + 1];

int d[n + 1];

memset(f, -1, sizeof(f)); // Khởi tạo mảng f với giá trị -1

memset(d, 0, sizeof(d)); // Khởi tạo mảng d với giá trị 0

f[0] = 0;

Mảng f được dùng để lưu trữ các chỉ số của các phần tử tạo thành tổng j. Mảng d dùng để đánh dấu phần tử thuộc nhóm nào. Khởi tạo f[0] bằng 0 vì tổng 0 có thể đạt được với không phần tử nào.

1. **Tính toán giá trị có thể đạt được**:

cpp

Copy code

for (int i = 1; i <= n; i++) {

for (int j = t; j >= a[i]; j--) {

if (f[j - a[i]] != -1 && f[j] == -1) {

f[j] = i;

}

}

}

Sử dụng phương pháp DP để xác định các giá trị tổng có thể đạt được từ các phần tử. Nếu tổng j - a[i] có thể đạt được và tổng j chưa được đạt, thì gán giá trị i cho f[j].

1. **Xác định tổng gần nhất với t/2**:

cpp

Copy code

int target = t;

while (f[target] == -1) target--;

Tìm tổng gần nhất với t mà có thể đạt được bằng cách giảm dần từ t cho đến khi tìm được một giá trị hợp lệ trong f.

1. **Xây dựng nhóm phần tử**:

cpp

Copy code

while (target > 0) {

d[f[target]] = 1;

target -= a[f[target]];

}

Dùng mảng f để xác định các phần tử thuộc nhóm đầu tiên.

1. **Tính toán và in kết quả**:

cpp

Copy code

long long s1 = 0, s2 = 0;

for (int i = 1; i <= n; i++) {

if (d[i] == 1) {

s1 += a[i];

} else {

s2 += a[i];

}

}

long long s = abs(s1 - s2);

cout << s << endl;

for (int i = 1; i <= n; i++) {

if (d[i] == 1) {

cout << a[i] << " ";

}

}

cout << endl;

for (int i = 1; i <= n; i++) {

if (d[i] != 1) {

cout << a[i] << " ";

}

}

cout << endl;

Tính tổng của các phần tử trong hai nhóm và in hiệu số của chúng. Sau đó, in các phần tử thuộc mỗi nhóm.

Code 2:

#include <bits/stdc++.h>

using namespace std;

int main()

{

// Sử dụng freopen để đọc từ tệp "chiakeo.inp" và ghi ra tệp "chiakeo.out"

freopen("chiakeo.inp", "r", stdin);

freopen("chiakeo.out", "w", stdout);

int n, t = 0;

cin >> n;

int a[n + 1];

for (int i = 1; i <= n; i++) {

cin >> a[i];

t += a[i];

}

t = t / 2;

int f[t + 1];

int d[n + 1];

memset(f, -1, sizeof(f)); // Khởi tạo mảng f với giá trị -1

memset(d, 0, sizeof(d)); // Khởi tạo mảng d với giá trị 0

f[0] = 0;

for (int i = 1; i <= n; i++) {

for (int j = t; j >= a[i]; j--) {

if (f[j - a[i]] != -1 && f[j] == -1) {

f[j] = i;

}

}

}

int target = t;

while (f[target] == -1) target--;

while (target > 0) {

d[f[target]] = 1;

target -= a[f[target]];

}

long long s1 = 0, s2 = 0;

for (int i = 1; i <= n; i++) {

if (d[i] == 1) {

s1 += a[i];

} else {

s2 += a[i];

}

}

long long s = abs(s1 - s2);

cout << s << endl;

for (int i = 1; i <= n; i++) {

if (d[i] == 1) {

cout << a[i] << " ";

}

}

cout << endl;

for (int i = 1; i <= n; i++) {

if (d[i] != 1) {

cout << a[i] << " ";

}

}

cout << endl;

return 0;

}

Code

#include <bits/stdc++.h>

using namespace std;

int main()

{

int n,t=0,f[10001],d[10001];

cin>>n;

int a[n+1];

for (int i=1;i<=n;i++)

{

cin>>a[i];

t=t+a[i];

}

t=t/2;

//QHD

for (int i=1;i<=t;i++)

{

f[i]=INT\_MAX;

for (int j=1;j<=n;j++)

if (i>=a[j] && j>f[i-a[j]])

{

f[i]=j;

break;

}

}

//Truyvet

while (f[t] > n) t--;

while (t > 0)

{

d[f[t]]=1;

t=t-a[f[t]];

}

long long s1=0; long long s2=0;

for (int i=1;i<=n;i++)

if (d[i]==1)

s1=s1+a[i];

else

s2=s2+a[i];

long long s=abs(s1-s2);

cout<<s<<endl;

for (int i=1;i<=n;i++)

if (d[i]==1) cout<<a[i]<<" ";

cout<<endl;

for (int i=1;i<=n;i++)

if (d[i]!=1) cout<<a[i]<<" ";

cout<<endl;

}

Chúng ta sẽ chạy tay từng bước mã nguồn này với bộ dữ liệu đầu vào mẫu. Dữ liệu đầu vào:

Copy code

7

7 9 14 18 15 3 7

### Bước 1: Đọc dữ liệu đầu vào

int n, t = 0, f[10001], d[10001];

cin >> n;

int a[n + 1];

for (int i = 1; i <= n; i++) {

cin >> a[i];

t = t + a[i];

}

* n = 7
* a = [0, 7, 9, 14, 18, 15, 3, 7] (a[0] không dùng)
* t = 7 + 9 + 14 + 18 + 15 + 3 + 7 = 73

### Bước 2: Tính t

t = t / 2;

* t = 73 / 2 = 36

### Bước 3: Khởi tạo bảng f và d

for (int i = 1; i <= t; i++) {

f[i] = INT\_MAX;

}

* f = [INT\_MAX, INT\_MAX, ..., INT\_MAX] (37 phần tử)

### Bước 4: Điền bảng f bằng Quy Hoạch Động

for (int i = 1; i <= t; i++) {

for (int j = 1; j <= n; j++) {

if (i >= a[j] && j > f[i - a[j]]) {

f[i] = j;

break;

}

}

}

#### Chi tiết các bước trong vòng lặp:

* Với i = 1 đến i = 36, kiểm tra từng gói kẹo j = 1 đến j = 7.

Ví dụ:

* Với i = 3 và j = 1:
  + i >= a[j] tức là 3 >= 7 không thỏa mãn. Do đó bỏ qua.
* Với i = 7 và j = 1:
  + i >= a[j] tức là 7 >= 7 thỏa mãn.
  + f[7 - 7] tức là f[0] = 0 (ban đầu f[0] = 0, không phải INT\_MAX).
  + 1 > f[0] thỏa mãn.
  + f[7] = 1.

Tiếp tục như vậy cho các giá trị i và j khác. Chúng ta sẽ có kết quả sau:

f = [0, INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX, INT\_MAX, 1, ..., 1, INT\_MAX]

(Ta bỏ qua các giá trị không thỏa mãn)

### Bước 5: Truy vết tìm giá trị t tốt nhất

while (f[t] > n) t--;

Vòng lặp này sẽ giảm giá trị t đến khi f[t] hợp lệ (nhỏ hơn hoặc bằng n).

### Bước 6: Truy vết để xác định các gói kẹo đã được chọn

while (t > 0) {

d[f[t]] = 1;

t = t - a[f[t]];

}

* d = [0, 0, 0, 0, 0, 0, 0, 0] (khởi tạo ban đầu)

#### Chi tiết các bước trong vòng lặp:

* Với t = 36:
  + d[f[36]] = d[1] = 1
  + t = t - a[1] = 36 - 7 = 29

Tiếp tục như vậy cho các giá trị t khác. Chúng ta sẽ có kết quả sau:

d = [0, 1, 1, 1, 0, 0, 1, 1]

### Bước 7: Tính tổng trọng lượng của hai phần và in kết quả

long long s1 = 0, s2 = 0;

for (int i = 1; i <= n; i++) {

if (d[i] == 1)

s1 += a[i];

else

s2 += a[i];

}

long long s = abs(s1 - s2);

cout << s << endl;

for (int i = 1; i <= n; i++) {

if (d[i] == 1) cout << a[i] << " ";

}

cout << endl;

for (int i = 1; i <= n; i++) {

if (d[i] != 1) cout << a[i] << " ";

}

cout << endl;

#### Tính toán cụ thể:

* s1 = 7 + 9 + 3 + 7 = 26
* s2 = 14 + 18 + 15 = 47
* s = abs(s1 - s2) = abs(26 - 47) = 21

### Kết quả đầu ra

21

7 9 3 7

14 18 15

**BÀI TOÁN: CHO THUÊ MÁY**

Tại thời điểm 0, ông chủ một máy tính hiệu năng cao nhận được đơn đặt hàng thuê sử dụng máy của n khách hàng. Các khách hàng được đánh số từ 1 đến n. Khách hàng i cần sử dụng máy từ thời điểm di đến thời điểm ci (di, ci là các số nguyên và 0 < di < ci < 1000000000) và sẽ trả tiền sử dụng máy là pi (pi nguyên, 0 < p i ≤ 10000000). Bạn cần xác định xem ông chủ cần nhận phục vụ những khách hàng nào sao cho khoảng thời gian sử dụng máy của hai khách được nhận phục vụ bất kỳ không được giao nhau đồng thời tổng tiền thu được từ việc phục vụ họ là lớn nhất.

***Dữ liệu vào:*** Từ file văn bản THUE.INP

Dòng đầu tiên ghi số n (0 < n =< 1000);

- Dòng thứ i+1 trong số n dòng tiếp theo ghi 3 số di, ci, pi cách nhau bởi dấu trắng (i = 1, 2,... n).

***Kết quả:***Ghi ra file văn bản THUE.OUT

-Dòng đầu tiên ghi hai số nguyên dương theo thứ tự là số lượng khách hàng nhận phục vụ và tổng tiền thu được từ việc phục vụ họ.

-Dòng tiếp theo ghi chỉ số của các khách hàng được nhận phục vụ.

Ví dụ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| THUE.INP | THUE.OUT |  | THUE.INP | THUE.OUT |
| 3  150 500 150  1 200 100  400 800 80 | 2 180  2 3 |  | 4  400 821 800  200 513 500  100 325 200  600 900 600 | 2 1100  2 4 |

Bài toán này chúng ta phải chú ý ở chỗ: Để dùng thuật toán Quy hoạch động tối ưu từng bước thì trước hết chúng ta phải sắp xếp các ci theo thứ tự tăng dần:

Giả sử c1≤ c2 ≤... ≤ cN.

Tương tự bài toán trên: Gọi F[k] là số tiền lớn nhất khi phục vụ một số khách hàng từ 1 đến k.

Với mỗi F[k] ta có:

- Nếu chấp nhận phục vụ khách k thì F[k]:=F[t]+pk (với t là chỉ số max thoả mãn khoảng thời gian [dt, ct [dk,ck] = ).

- Nếu không chấp nhận phục vụ k thì F[k]:=F[k-1].

Như vậy hàm quy hoạch động của F[k] sẽ là:

F[k]:=Max{F[t]+pk,F[k-1]} với k = 2, 3,... N và t có ý nghĩa như trên.

#include <bits/stdc++.h>

using namespace std;

const int N = 1000;

const int INF = int(1e9);

struct don {

int l, r, v, id;

};

don a[N + 1];

int dp[N + 1], vet[N + 1];

vector<int> res;

int n;

// Hàm so sánh để sắp xếp các công việc theo thời gian kết thúc

bool cmp(don x, don y) {

return x.r < y.r;

}

int main() {

cin >> n;

for (int i = 1; i <= n; i++) {

cin >> a[i].l>> a[i].r >> a[i].v;

a[i].id = i;

}

// Sắp xếp các công việc theo thời gian kết thúc

sort(a + 1, a + n+ 1, cmp);

// Khởi tạo mảng dp

fill(dp, dp + n + 1, 0);

// Tính giá trị lớn nhất sử dụng lập trình động

for (int i = 1; i <= n; i++) {

for (int j = 0; j < i; j++) {

if (a[j].r<= a[i].l && dp[i] < dp[j] + a[i].v) {

dp[i] = dp[j] + a[i].v;

vet[i] = j;

}

}

}

// Tìm giá trị lớn nhất trong dp

int maxgt = 0, maxss = 0;

for (int i = 1; i <= n; i++) {

if (dp[i] > maxgt) {

maxgt = dp[i];

maxss = i;

}

}

// Truy vết để tìm các công việc được chọn

while (maxss != 0) {

res.push\_back(a[maxss].id);

maxss = vet[maxss];

}

// Xuất kết quả

cout << res.size() << " " << maxgt << endl;

for (int i = res.size() - 1; i >= 0; i--) {

cout << res[i] << " ";

}

return 0;

}

**Đầu vào**

Copy code

4

400 821 800

200 513 500

100 325 200

600 900 600

**Bước 1: Đọc dữ liệu đầu vào**

cpp

Copy code

cin >> n; // n = 4

for (int i = 1; i <= n; i++) {

cin >> a[i].l >> a[i].r >> a[i].v;

a[i].id = i;

}

Dữ liệu sau khi đọc:

* a[1] = {400, 821, 800, 1}
* a[2] = {200, 513, 500, 2}
* a[3] = {100, 325, 200, 3}
* a[4] = {600, 900, 600, 4}

**Bước 2: Sắp xếp các công việc dựa trên thời gian kết thúc**

cpp

Copy code

sort(a + 1, a + n + 1, cmp);

Sau khi sắp xếp:

* a[1] = {100, 325, 200, 3}
* a[2] = {200, 513, 500, 2}
* a[3] = {400, 821, 800, 1}
* a[4] = {600, 900, 600, 4}

**Bước 3: Khởi tạo các phần tử giả**

cpp

Copy code

a[0] = {-inf, -inf, 0, 0};

a[n + 1] = {inf, inf, 0, n + 1};

**Bước 4: Khởi tạo mảng DP và Vet**

cpp

Copy code

for (int i = 2; i <= n + 1; i++) {

for (int j = 0; j < i; j++) {

if (a[j].r <= a[i].l && dp[i] < dp[j] + a[i].v) {

dp[i] = dp[j] + a[i].v;

vet[i] = j;

}

}

}

**Đầu vào**

Copy code

4

400 821 800

200 513 500

100 325 200

600 900 600

**Bước 1: Đọc dữ liệu và sắp xếp**

* **Dữ liệu sau khi đọc**:
  + a[1] = {400, 821, 800, 1}
  + a[2] = {200, 513, 500, 2}
  + a[3] = {100, 325, 200, 3}
  + a[4] = {600, 900, 600, 4}
* **Sắp xếp các công việc dựa trên thời gian kết thúc**:
  + a[1] = {100, 325, 200, 3}
  + a[2] = {200, 513, 500, 2}
  + a[3] = {400, 821, 800, 1}
  + a[4] = {600, 900, 600, 4}

**Bước 2: Khởi tạo mảng dp và vet**

cpp

Copy code

for (int i = 1; i <= n + 1; i++) {

dp[i] = 0; // Giá trị tối đa ban đầu là 0

vet[i] = 0; // Không có công việc trước đó

}

* **Giá trị ban đầu**:
  + dp = [0, 0, 0, 0, 0, 0]
  + vet = [0, 0, 0, 0, 0, 0]

**Bước 3: Tính toán cho từng công việc**

**Tính toán cho i = 1** (công việc a[1]):

* Không có công việc nào trước a[1] nên dp[1] = a[1].v = 200:
  + dp[1] = 200
  + vet[1] = 0

**Tính toán cho i = 2** (công việc a[2]):

* **j = 0** (công việc giả định):
  + a[0].r <= a[2].l là đúng vì -inf <= 200.
  + dp[2] = dp[0] + a[2].v = 0 + 500 = 500
  + vet[2] = 0
* **j = 1** (công việc a[1]):
  + a[1].r (325) không nhỏ hơn hoặc bằng a[2].l (200), không thay đổi.
  + dp[2] không thay đổi.
* **Kết quả**:
  + dp[2] = 500
  + vet[2] = 0

**Tính toán cho i = 3** (công việc a[3]):

* **j = 0** (công việc giả định):
  + a[0].r <= a[3].l là đúng vì -inf <= 400.
  + dp[3] = dp[0] + a[3].v = 0 + 800 = 800
  + vet[3] = 0
* **j = 1** (công việc a[1]):
  + a[1].r (325) nhỏ hơn a[3].l (400), cập nhật giá trị.
  + dp[3] = dp[1] + a[3].v = 200 + 800 = 1000
  + vet[3] = 1
* **j = 2** (công việc a[2]):
  + a[2].r (513) không nhỏ hơn a[3].l (400), không thay đổi.
  + dp[3] không thay đổi.
* **Kết quả**:
  + dp[3] = 1000
  + vet[3] = 1

**Tính toán cho i = 4** (công việc a[4]):

* **j = 0** (công việc giả định):
  + a[0].r <= a[4].l là đúng vì -inf <= 600.
  + dp[4] = dp[0] + a[4].v = 0 + 600 = 600
  + vet[4] = 0
* **j = 1** (công việc a[1]):
  + a[1].r (325) nhỏ hơn a[4].l (600), cập nhật giá trị.
  + dp[4] = dp[1] + a[4].v = 200 + 600 = 800
  + vet[4] = 1
* **j = 2** (công việc a[2]):
  + a[2].r (513) nhỏ hơn a[4].l (600), cập nhật giá trị.
  + dp[4] = dp[2] + a[4].v = 500 + 600 = 1100
  + vet[4] = 2
* **j = 3** (công việc a[3]):
  + a[3].r (821) không nhỏ hơn a[4].l (600), không thay đổi.
  + dp[4] không thay đổi.
* **Kết quả**:
  + dp[4] = 1100
  + vet[4] = 2

**Kết quả cuối cùng**

cpp

Copy code

cout << res.size() - 1 << " " << dp[n + 1] << endl;

for (int i = res.size() - 2; i >= 0; i--) {

cout << res[i] << " ";

}

**Truy vết ngược**:

* Bắt đầu từ j = 5:
  + res.push\_back(a[4].id) => res = [4]
  + j = vet[4] = 2
  + res.push\_back(a[2].id) => res = [4, 2]
  + j = vet[2] = 0
  + res.push\_back(a[0].id) => res = [4, 2, 0]
* **Kết quả in ra**:
  + Số lượng công việc được chọn: 2
  + Giá trị tối đa đạt được: 1100
  + Danh sách các công việc được chọn: 2 và 4

THAM QUAN

* VMschools là hệ thống trường song ngữ đào tạo liên thông từ cấp Tiểu học đến Trung học phổ thông trong một môi trường giáo dục toàn diện "Đức - Trí - Thể - Mỹ".
* Nhân dịp kỉ niệm 10 năm ngày thành lập trường, trường muốn tổ chức cho các em học sinh đi tham quan tại một số địa điểm.
* Trường có m khối, mỗi khối có n lớp, mỗi lớp có k (1 < k < 200) học sinh đăng kí tham gia.
* Ban đại diện cha mẹ học sinh của trường, của các lớp đều có cử ít nhất một người tham gia để hỗ trợ nhà trường quản lý các em học sinh nhằm đảm bảo an toàn cho chuyến đi.
* Căn cứ vào số lượng học sinh đăng ký tham gia, trường đã hợp đồng hai chiếc xe du lịch loại lớn để phục vụ tất cả các em.

**Yêu cầu:** Hãy phân chia, bố trí học sinh giữa các khối, các lớp ngồi ở hai xe phù hợp để số lượng học sinh giữa hai xe chênh lệch nhau ít nhất và cho biết mỗi xe sẽ chở bao nhiêu em?

**Dữ liệu vào:** Cho bởi file VISIT.INP bao gồm:

* Dòng thứ nhất chứa 2 số m, n (1 ≤ m ≤ 15, 1 ≤ n ≤ 20) lần lượt là số khối và số lớp trong mỗi khối.
* m dòng tiếp theo, mỗi dòng gồm n số thể hiện số học sinh mỗi lớp đăng ký tham gia chuyến đi, các số cách nhau một khoảng trắng.

**Kết quả:** Ghi ra file VISIT.OUT bao gồm:

* m dòng đầu, mỗi dòng gồm n số 1 và 0 kí hiệu xe thứ nhất, xe thứ hai sẽ phục vụ các em học sinh của các khối lớp tương ứng đã đăng ký tham gia.
* Dòng cuối cùng ghi lần lượt tổng số học sinh sẽ ngồi lên xe thứ nhất và xe thứ hai.
* **Ví dụ:**

|  | **VISIT.INP** | **VISIT.OUT** |
| --- | --- | --- |
|  | 4 3 | 1 0 0 |
|  | 12 0 9 | 1 1 1 |
|  | 4 10 3 | 0 0 1 |
|  | 12 8 0 | 0 0 1 |
|  | 0 7 6 |  |
|  |  | 35 36 |

5 6

50 46 16 0 0 0

20 51 23 5 0 0

20 12 2 5 0 0

10 19 12 10 0 0

0 0 0 0 0 0

#include <bits/stdc++.h>

using namespace std;

int main() {

//freopen("VISIT.INP", "r", stdin);

//freopen("VISIT.OUT", "w", stdout);

int N, M, i, j, k; // N: số lượng học sinh, M: số lượng xe

cin >> N >> M;

int a[10000][10000], c[10000][10000] = {0}; // Khởi tạo mảng c với giá trị ban đầu là 0

for (i = 1; i <= N; ++i)

for (j = 1; j <= M; ++j)

cin >> a[i][j];

int max\_a = 0;

for (i = 1; i <= N; ++i) {

for (j = 1; j <= M; ++j) {

if (a[i][j] > max\_a) {

max\_a = a[i][j];

}

}

}

// Khởi tạo mảng B

int max\_size = 400 \* max\_a;

int B[4][40000] = {0}; // Khởi tạo mảng B với giá trị ban đầu là 0

int tong = 0;

// Khởi tạo giá trị đầu tiên của B để bắt đầu chuỗi tính toán

B[1][0] = 1; // Đặt 1 ở vị trí ban đầu

for (i = 1; i <= N; ++i) {

for (j = 1; j <= M; ++j) {

for (k = tong; k >= 0; --k) {

if (B[1][k] > 0 && B[1][k + a[i][j]] == 0) {

B[1][k + a[i][j]] = 1;

B[2][k + a[i][j]] = i;

B[3][k + a[i][j]] = j;

}

}

tong += a[i][j];

}

}

int u = tong;

tong = tong / 2;

while (B[1][tong] == 0 && tong > 0) tong--;

while (tong > 0) {

c[B[2][tong]][B[3][tong]] = 1;

tong -= a[B[2][tong]][B[3][tong]];

}

// In mảng c

cout << "Mảng c (các phần tử thuộc xe 1):" << endl;

for (i = 1; i <= N; ++i) {

for (j = 1; j <= M; ++j) {

cout << c[i][j] << " ";

}

cout << endl;

}

// Tính tổng các chỉ số của hai xe và in độ lệch

int sum1 = 0, sum2 = 0;

for (i = 1; i <= N; ++i) {

for (j = 1; j <= M; ++j) {

if (c[i][j] == 1) {

sum1 += a[i][j];

} else {

sum2 += a[i][j];

}

}

}

int chenh\_lech = abs(sum1 - sum2);

cout << "Tổng chỉ số xe 1: " << sum1 << endl;

cout << "Tổng chỉ số xe 2: " << sum2 << endl;

cout << "Độ lệch giữa hai xe: " << chenh\_lech << endl;

return 0;

}

#include <bits/stdc++.h>

using namespace std;

int main() {

ifstream infile("VISIT.INP");

ofstream outfile("VISIT.OUT");

int m, n;

infile >> m >> n;

vector<vector<int>> students(m, vector<int>(n));

vector<vector<int>> allocation(m, vector<int>(n));

int total\_students = 0;

// Đọc dữ liệu

for (int i = 0; i < m; ++i) {

for (int j = 0; j < n; ++j) {

infile >> students[i][j];

total\_students += students[i][j];

}

}

// Tổng số học sinh cần phân chia

int half = total\_students / 2;

// DP array to track achievable sums

vector<bool> dp(half + 1, false);

vector<vector<bool>> keep(m, vector<bool>(n, false));

dp[0] = true;

// Perform subset-sum DP

for (int i = 0; i < m; ++i) {

for (int j = 0; j < n; ++j) {

int current\_students = students[i][j];

for (int k = half; k >= current\_students; --k) {

if (dp[k - current\_students]) {

dp[k] = true;

keep[i][j] = true;

}

}

}

}

// Determine closest achievable sum to half

int sum1 = 0;

for (int i = half; i >= 0; --i) {

if (dp[i]) {

sum1 = i;

break;

}

}

int sum2 = total\_students - sum1;

// Allocate students to each bus

int sum1\_current = 0;

for (int i = 0; i < m; ++i) {

for (int j = 0; j < n; ++j) {

if (sum1\_current + students[i][j] <= sum1) {

allocation[i][j] = 1;

sum1\_current += students[i][j];

} else {

allocation[i][j] = 0;

}

}

}

// Ghi kết quả vào file

for (int i = 0; i < m; ++i) {

for (int j = 0; j < n; ++j) {

outfile << allocation[i][j] << " ";

}

outfile << endl;

}

outfile << sum1 << " " << sum2 << endl;

return 0;

}

Code 3:

#include <bits/stdc++.h>

using namespace std;

int main() {

int N, M; // N: số lượng học sinh, M: số lượng xe

cin >> N >> M;

vector<vector<int>> a(N + 1, vector<int>(M + 1)); // Khởi tạo mảng a với kích thước N x M

vector<vector<int>> c(N + 1, vector<int>(M + 1, 0)); // Khởi tạo mảng c với giá trị ban đầu là 0

int max\_a = 0;

for (int i = 1; i <= N; ++i) {

for (int j = 1; j <= M; ++j) {

cin >> a[i][j];

if (a[i][j] > max\_a) {

max\_a = a[i][j];

}

}

}

// Khởi tạo mảng B

int max\_size = 400 \* max\_a;

vector<vector<int>> B(4, vector<int>(max\_size + 1, 0)); // Khởi tạo mảng B với giá trị ban đầu là 0

int tong = 0;

// Tính toán khả năng đạt được tổng các phần tử

for (int i = 1; i <= N; ++i) {

for (int j = 1; j <= M; ++j) {

for (int k = tong; k >= 1; --k) {

if (B[1][k] > 0 && B[1][k + a[i][j]] == 0) {

B[1][k + a[i][j]] = 1;

B[2][k + a[i][j]] = i;

B[3][k + a[i][j]] = j;

}

}

tong += a[i][j];

if (B[1][a[i][j]] == 0) {

B[1][a[i][j]] = 1;

B[2][a[i][j]] = i;

B[3][a[i][j]] = j;

}

}

}

int u = tong;

u = u / 2;

while (B[1][u] == 0) u--;

while (u > 0) {

c[B[2][u]][B[3][u]] = 1;

u -= a[B[2][u]][B[3][u]];

}

// In mảng c

cout << "Mảng c (các phần tử thuộc xe 1):" << endl;

for (int i = 1; i <= N; ++i) {

for (int j = 1; j <= M; ++j) {

cout << c[i][j] << " ";

}

cout << endl;

}

// Tính tổng các chỉ số của hai xe và in độ lệch

int sum1 = 0, sum2 = 0;

for (int i = 1; i <= N; ++i) {

for (int j = 1; j <= M; ++j) {

if (c[i][j] == 1) {

sum1 += a[i][j];

} else {

sum2 += a[i][j];

}

}

}

int chenh\_lech = abs(sum1 - sum2);

cout << "Tổng chỉ số xe 1: " << sum1 << endl;

cout << "Tổng chỉ số xe 2: " << sum2 << endl;

cout << "Độ lệch giữa hai xe: " << chenh\_lech << endl;

return 0;

}