Bài **DEADLINES**

Solutions:

Sử dụng chiến lược tham lam để sắp xếp các bài tập sẽ làm để hoàn thành tất cả đúng hạn. Đầu tiên, Ta thấy rằng vào ngày N, chúng ta chỉ có thể hoàn thành các bài tập đến hạn vào ngày N. Nếu có bất kỳ bài tập nào như vậy tồn tại, chúng ta có thể tham lam thực hiện bài tập ngoài cùng bên phải vì nó yêu cầu số lần hoán đổi ít nhất để chuyển sang ngày N. Vào ngày N-1, chúng ta có thể hoàn thành các bài tập với thời hạn là ngày N- 1 hoặc N. Một lần nữa chúng ta có thể tham lam thực hiện bài tập đúng nhất như vậy.

Thuật toán tham lam ở đây là lặp các ngày từ N đến 1 và vào mỗi ngày, thực hiện nhiệm vụ ngoài cùng bên phải đến hạn vào hoặc sau ngày hiện tại.

Để có được tổng số lần hoán đổi, còn được gọi là số lần đảo ngược, ta sẽ sử cây Fenwick-Tree để tính.

Nếu vào bất kỳ ngày thứ i nào, không có bài tập nào chúng ta có thể hoàn thành thì chúng ta sẽ cần phải hoàn thành i bài tập còn lại trong i-1 ngày, điều này là không thể.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int n;
int D[200001];
vector<int> L[200001];
int R[200001];
int fen[200001];
//cap nhat cay Fenwick-Tree
void update(int i) {
    while (i <= n) {
        ++fen[i];
        i += i & -i;
    }
}
//Tinh tong so lan hoan doi
int query(int i) {
    int ret = 0;
    while (i) {
        ret += fen[i];
        i -= i & -i;
```

```
    }
    return ret;
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; ++i) cin >> D[i],
L[D[i]].push_back(i);
    priority_queue<int> pq;
    long long ans = 0;
    for (int i = n; i > 0; --i) {
        for (int j : L[i]) pq.push(j);
        if (pq.empty()) {
            cout<<"-1\n";
            return 0;
        }
        R[i] = pq.top(); pq.pop();
        ans += query(R[i]);
        update(R[i]);
    }
    cout<< ans;
    return 0;
}
```

Bài 2. GYM

Solutions:

Tham lam: sắp xếp dãy tăng dần theo xi+li.

Sử dụng queue ưu tiên: xét lần lượt các phòng tập từ i đến n.

Level+=xi;

Đưa xi và queue.

Nếu level>xi+li thì cần loại bỏ giá trị lớn nhất đang có trong queue.

Lưu ý tính lại level-=queue.top().

Loại queue.top cho đến khi nào level<=xi+li.

Lặp lại cho đến hết ta có số lượng còn lại trong queue là số lượng các phòng tập có thê tham gia.

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
#define mp make_pair
#define fi first
#define se second
int n;
pair<long long, long long> a[N];

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    cin >> n;
    for (int i = 1; i <= n; ++i)
        cin >> a[i].fi;
    for (int i = 1; i <= n; ++i)
        cin >> a[i].se;

    sort(a + 1, a + n + 1, [&] (pair<long long, long
long> x, pair<long long, long long> y)
     {        return x.fi + x.se < y.fi + y.se;    } );

    priority_queue<long long> q;
    int level = 0, res = 0;
    for (int i = 1; i <= n; ++i) {
        level += a[i].fi;
        q.push(a[i].fi);
        while (level > a[i].fi + a[i].se) {
            level -= q.top();
            //cout<<i<<" i=  \n";
            q.pop();
        }
        res = max(res, (int) q.size());
    }
```

```
    cout << res;

    return 0;
}
```

Bài 3. QUALUUNIEM

Solution:

Tham lam, khi chiếm cùng một không gian, phải chọn cái có giá trị nhất. Gọi dp[i][j] là giá trị thu được khi chọn loại i với khối lượng tối đa là j.

Đầu tiên chia thành 3 loại theo khối lượng 1, 2, 3 rồi sắp xếp theo thứ tự giảm dần, đầu tiên chỉ lấy sản phẩm có giá 1 và 2 cho DP.

Sau đó cố gắng thay thế mặt hàng 1 và 2 bằng mặt hàng có giá 3 để tối đa hóa tổng giá.

Đầu tiên ta tính bài toán đó với ưu tiên chỉ chọn loại 1 kg.

Khi đó:

dp[1][j]=max(dp[1][j-1]+q[1].top().

Dùng queue ưu tiên vì nếu cùng khối lượng ta phải chọn cái giá trị hơn. (i=1..n1 và j<=w là số lượng đồ vật loại 1).

Tính dp[2][j] là giá trị thu được khi chọn các đồ vật loại 2 với khối lượng tối đa là j.

Tương tự như tính loại 1. Tuy nhiên ta sẽ **thử thay thế hai đồ vật loại 1 lấy 1 đồ vật loại 2** được không?

Nếu thay được thì tính toán cập nhật lại dp[2][j]=dp[1][j-2*t-2]+c+sum; với t là số lần đã thay loại 1 bằng loại 2, c là đồ vật loại 2 tốt nhất hiện tại, sum là tổng giá trị đã thay thế được.

Nếu không thay thế được thì dp[2][j]=dp[1][j-2*t]+sum;

Nhớ tính phần còn dư khi đã hết đồ vật loại 2.

dp[2][j]=dp[1][j-2*t]+sum; (với j từ khối lượng chưa tính đến w).

Tính dp[3][w] trực tiếp:

Dp[3][w]=dp[2][w];

Sau đó thử thay các đồ vật loại 3 cho đồ vật loại 2 trước đó.

Với t là khối lượng đã thay bằng loại 3. Xét các đồ vật loại 3 ta có.

```
while(q[3].size())
    {
        t=t+3;
        sum=sum+q[3].top();
        q[3].pop();
        if(t<=w) dp[3][w]=max(dp[3][w],dp[2][w-t]+sum);
    }
```

Kết quả: dp[3][w].

Code:

```
#include<bits/stdc++.h>
using namespace std;
int n,w;
long long dp[4][300005];
priority_queue<int> q[4];
void chonloai1()
{
    int k=q[1].size();
    for(int j=1;j<=k && j<=w;j++)
        {
            dp[1][j]=dp[1][j-1]+q[1].top();
            q[1].pop();
        }
    for(int i=k+1;i<=w;i++)
        dp[1][i]=dp[1][i-1];

}
void chonloai2()
{
    int k=q[2].size();
    long long sum=0, t=0, c=0, l=1;
```

```
     for(int i=1;i<=k && i<=w;i++)
     {
          c=q[2].top();
          q[2].pop();
          for(int j=l;j<=w;j++)
          {
               l=j;
     if(j>=2*(t+1) && dp[1][j-2*t]+sum<dp[1][j-2*t-2]+c+sum)
               {
                    dp[2][j]=dp[1][j-2*t-2]+c+sum;
                    sum=sum+c;
                    t=t+1;
                    break;
               }
               else
               {
                    dp[2][j]=dp[1][j-2*t]+sum;
               }
          }

     }
     for(int i=l;i<=w;i++)
          dp[2][i]=dp[1][i-2*t]+sum;
}
void chonloai3()
{
     dp[3][w]=dp[2][w];
     long long t=0,sum=0;
     while(q[3].size())
     {
          t=t+3;
          sum=sum+q[3].top();
          q[3].pop();
          if(t<=w) dp[3][w]=max(dp[3][w],dp[2][w-t]+sum);
     }
}

int main()
{
     freopen("qualuuniem.inp","r",stdin);
    freopen("qualuuniem.out","w",stdout);
     cin>>n>>w;
     int ww,c;
     for(int i=1;i<=n;i++)
```

```
    {
        cin>>ww>>c;
        if(ww==1) q[1].push(c);
        if(ww==2) q[2].push(c);
        if(ww==3) q[3].push(c);
    }
    chonloai1();
    chonloai2();
    chonloai3();
    cout<<dp[3][w];
    return 0;
}
```

Bài 4. GAMESO

Solutions:

Gọi dp[i][val] ghi nhận có thể tạo ra giá trị val bắt đầu từ chỉ số i hay không. Ban đầu dp[i][val] tất cả bằng -1. Ghi nhận không thể tạo giá trị val từ vị trí i.

Ta thấy dp[i][nums[i]]=i. vì chúng ta có thể tạo ra số nums[i] ghi nhận giá trị của dp.

Với dữ liệu từ 1 đến 40 thì kết quả tối đa có thể tạo khoảng 80. Ta thử xem số tối đa có thể tạo là bao nhiêu với val từ 1 tới 80.

Bắt đầu từ vị trí i=n-1➔0

Ta thấy dp[i][val] có thể tạo nếu dp[dp[i][val-1]+1][val-1] đã được tạo.

Nếu tạo được thì ghi nhận kích thước maxsize=max(maxsize,val).

Xuất kết quả.

Code:

```
#include<bits/stdc++.h>
using namespace std;
vector<int> nums;
int n;
int main()
{
    cin>>n;
    nums.resize(n);
```

```
    for(int i=0;i<n;i++)
        cin>>nums[i];
    int siz=80;
    int maxsize=0;
    //khoi tao dp ban dau tat ca la -1.
    vector<vector<int>>dp(n,vector<int> (siz,-1));
    for(int i=0;i<n;i++)
        dp[i][nums[i]]=i;
        //tao duoc gia tri nums[i]
    for(int val=1;val<siz;val++)
    {
        for(int i=n-1;i>=0;i--)
        {
            if(dp[i][val-1]==-1|| dp[i][val-1]+1>=n||
val-1<0)
    continue;//loai cac truong hop khong hop le
            dp[i][val]=dp[dp[i][val-1]+1][val-1];
            if(dp[i][val]!=-1)
maxsize=max(maxsize,val);
        }
    }
    cout<<maxsize;
}
```