DIVIDE AND CONQUER DP

Chia để trị (divide and conquer) là một phương pháp tối ưu việc tính công thức qui hoạch động.

Điều kiện để thực hiện

Một vài bài toán qui hoạch động đưa về việc tính công thức dạng:

$$dp(i,j) = \min_{0 \le k \le j} \{dp(i-1,k-1) + C(k,j)\}$$

Ở đây $C(k,j)$ là hàm chi phí và $dp(i,j) = 0$ khi $j < 0$.

Giả sử $0 \le i < m$ và $0 \le j < n$ và hàm chi phí có thời gian tính là O(1). Việc tính đơn giản công thức qui hoạch động trên có thời gian là $O(mn^2)$ vì có $m \times n$ trạng thái và n trạng thái chuyển. Đặt $opt(i,j)$ là giá trị k làm biểu thức trên đạt giá trị nhỏ nhất. Nếu $opt(i,j) \le opt(i,j+1)$ với mọi i,j chúng ta có thể sử dụng kỹ thuật chia để trị. Đây được gọi là *điều kiện đơn điệu*: Điểm phân tách tối ưu khi i cố định sẽ tăng khi j tăng.

Ta sẽ tính công thức qui hoạch động hiệu quả hơn. Giả sử đã xác định được $opt(i,j)$ khi cố định i và j. Khi đó với bất kỳ $j' < j$ ta biết rằng $opt(i,j') \le opt(i,j)$. Điều này có nghĩa là khi tính $opt(i,j')$ ta không phải kiểm tra nhiều vị trí cho vị trí đạt min (điểm chia tách).

Để giảm thiểu thời gian chạy chúng ta áp dụng kỹ thuật chia để trị. Trước tiên xác định $opt(i,n/2)$, sau đó tính $opt(i,n/4)$ biết rằng nó nhỏ hơn hoặc bằng $opt(i,n/2)$ và $opt(i,3n/4)$ biết rằng nó lớn hơn hoặc bằng $opt(i,n/2)$. Bằng cách theo dõi các giới hạn trên và giới hạn dưới của lựa chọn điểm chia chúng ta đạt được thời gian $O(mn\log n)$. Mỗi giá trị có thể có của $opt(i,j)$ chỉ xuất hiện tối đa trong $\log n$ lần trong quá trình tính.

Lưu ý rằng $opt(i,j)$ "cân bằng" như thế nào không quan trọng. Trên một mức cố định, giá trị k được sử dụng nhiều nhất 2 lần và có $\log n$ mức như vậy.

Triển khai chung

Mặc dù có thể triển khai theo nhiều cách khác nhau tùy theo từng bài nhưng dưới đây là mẫu chung: Hàm **compute** xác định hàng i lưu vào mảng **dp_cur**. Trạng thái của hàng $i-1$ được lưu trong **dp_before**. Nó được gọi đệ qui ban đầu **compute(0,n-1,0,n-1)**. Hàm xác định m hàng và trả về kết quả.

```
int m, n;
vector<long long> dp_before(n), dp_cur(n);

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid), k});
```

```
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int solve() {
    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```

Chú ý:

Khó khăn lớn nhất của việc sử dụng kỹ thuật chia để trị là chúng minh tính đơn điểu của hàm $opt(i,j)$. Nhiều vấn đề chia để trị cũng có thể được giải quyết bằng thủ thuật lồi (convex hull trick) hoặc ngược lại. Rất hữu ích nếu biết cả hai.