Thuật toán ứng dụng
Bài thực hành số 4: Quy hoạch động

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương

Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 16 tháng 5 năm 2021

Mục lục

1. MỞ ĐẦU
2. GOLD MINING
3. WAREHOUSE
4. RETAIL OUTLETS
5. MACHINE
6. TỔNG KẾT

Mục lục

1. MỞ ĐẦU
2. GOLD MINING
3. WAREHOUSE
4. RETAIL OUTLETS
5. MACHINE
6. TỔNG KẾT

Quy hoạch động là gì?

- Quy hoạch động là một mô hình giải bài.
- Ý tưởng là giải lần lượt các bài toán con, kích thước tăng dần, trong đó lời giải của một bài toán phụ thuộc vào lời giải của các bài toán con nhỏ hơn để thu được lời giải của bài toán ban đầu.
- Có thể cài đặt bằng hàm đệ quy hoặc vòng lặp.

MỞ ĐẦU

Quy hoạch động là gì?

- Quy hoạch động là một mô hình giải bài.
- Ý tưởng là giải lần lượt các bài toán con, kích thước tăng dần, trong đó lời giải của một bài toán phụ thuộc vào lời giải của các bài toán con nhỏ hơn để thu được lời giải của bài toán ban đầu.
- Có thể cài đặt bằng hàm đệ quy hoặc vòng lặp.

Quy hoạch động và **Chia để trị** có gì khác nhau?

Nhắc lại về **Chia để trị**:

- Chia để trị là một mô hình giải bài.
- Ý tưởng là chia bài toán lớn thành các bài toán con độc lập, giải từng bài toán con rồi kết hợp lời giải để thu được lời giải của bài toán ban đầu.
- Thường được cài đặt bằng hàm đệ quy.

Mô hình	**Quy hoạch động**	**Chia để trị**
Bài toán con	"Gối nhau", kích thước tăng dần	Độc lập, thường có kích thước bằng nhau
Cài đặt	Hàm đệ quy hoặc Vòng lặp	Thường dùng hàm đệ quy
Song song hóa	Không thể	Có thể

Bảng 1: Sự khác nhau giữa mô hình Quy hoạch động và mô hình Chia để trị

Mục lục

1 MỞ ĐẦU

2 GOLD MINING

3 WAREHOUSE

4 RETAIL OUTLETS

5 MACHINE

6 TỔNG KẾT

05. GOLD MINING

- Có n nhà kho nằm trên một đoạn thẳng.
- Nhà kho i có toạ độ là i và chứa lượng vàng là a_i.
- **Yêu cầu:** Chọn một số nhà kho sao cho:
  - Tổng lượng vàng lớn nhất.
  - 2 nhà kho liên tiếp có khoảng cách nằm trong đoạn $[L_1, L_2]$.
- Giới hạn:
  - $1 \leq n \leq 100000$.
  - $1 \leq L_1 \leq L_2 \leq n$.

Quy hoạch động

Quy hoạch động

- Bài toán con: $F[i]$ là tổng số vàng nhận được nếu nhà kho i là nhà kho cuối cùng được chọn.

Quy hoạch động

- Bài toán con: $F[i]$ là tổng số vàng nhận được nếu nhà kho i là nhà kho cuối cùng được chọn.
- Kết quả: $max_i F[i]$.

Quy hoạch động

- Bài toán con: $F[i]$ là tổng số vàng nhận được nếu nhà kho i là nhà kho cuối cùng được chọn.
- Kết quả: $max_i F[i]$.
- Khởi tạo: $F[i] = a[i], \forall i < L_1$.

Quy hoạch động

- Bài toán con: $F[i]$ là tổng số vàng nhận được nếu nhà kho i là nhà kho cuối cùng được chọn.
- Kết quả: $max_i F[i]$.
- Khởi tạo: $F[i] = a[i], \forall i < L_1$.
- Công thức phụ thuộc:

$$F[i] = max_{j \in [i-L_2, i-L_1]}(a[i] + F[j]), \forall i \in [L_1, n) \quad (1)$$

Tìm kiếm vét cạn 1a

Tìm kiếm vét cạn:

- Nhà kho thứ i có thể được chọn hoặc không $\rightarrow$ có 2^n cách chọn.
- Với mỗi cách chọn, kiểm tra xem 2 nhà kho liên tiếp $i, j (i < j)$ có thoả mãn $L_1 <= j - i <= L_2$ không, nếu thoả mãn thì tính tổng số vàng và cập nhật kết quả tốt nhất.
- Có thể sử dụng stack để lưu danh sách các nhà kho được chọn.
- Độ phức tạp: $O(2^n \times n)$.

Code 1a

```
void _try(int x) {
    if (x == n) {
        updateResult();
    }
    _try(x + 1);
    s.push(x);
    _try(x + 1);
    s.pop();
}

void main() {
    try(0);
}
```

Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.

Tìm kiếm vét cạn 1b

Nhận xét:

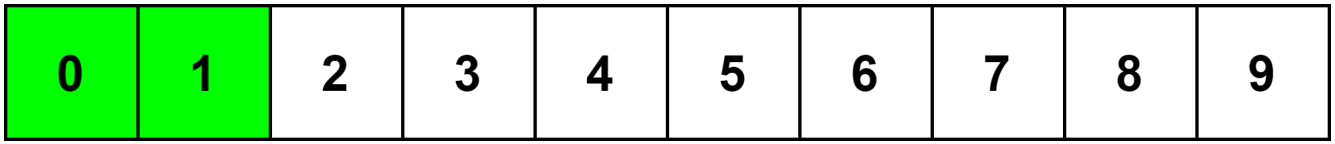
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:

0	1	2	3	4	5	6	7	8	9

Tìm kiếm vét cạn 1b

Nhận xét:

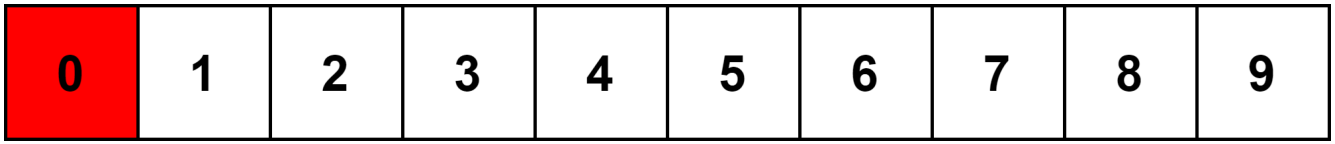
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

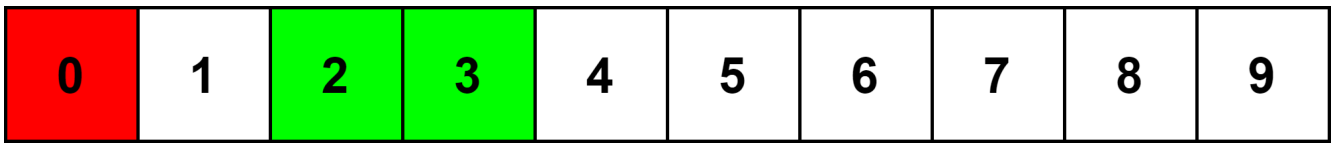
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

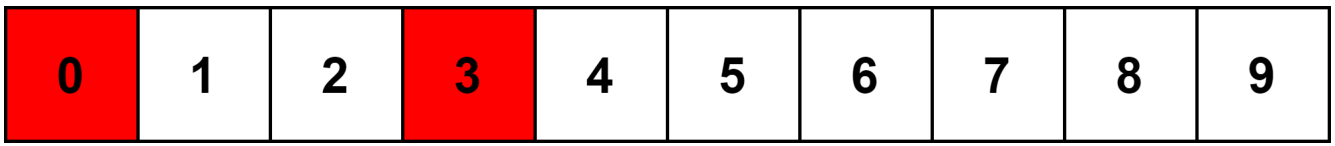
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

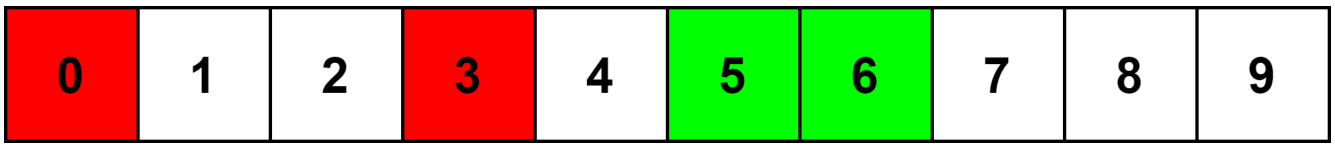
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

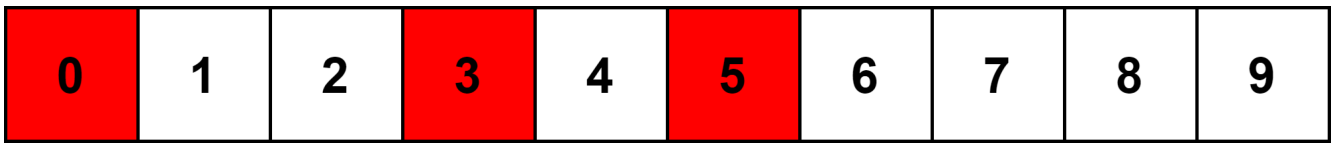
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

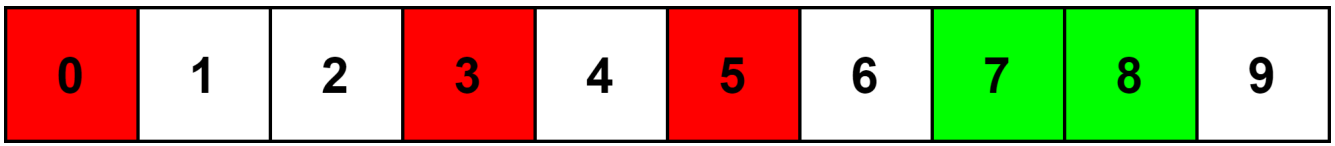
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

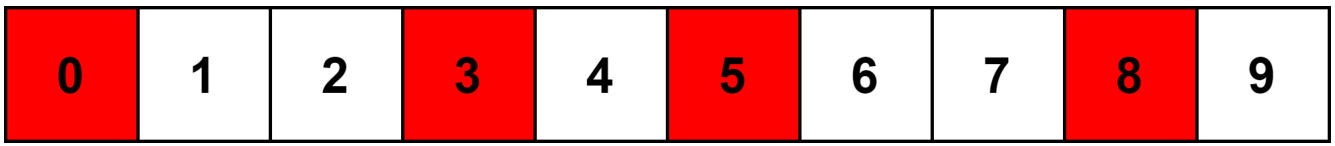
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



Tìm kiếm vét cạn 1b

Nhận xét:

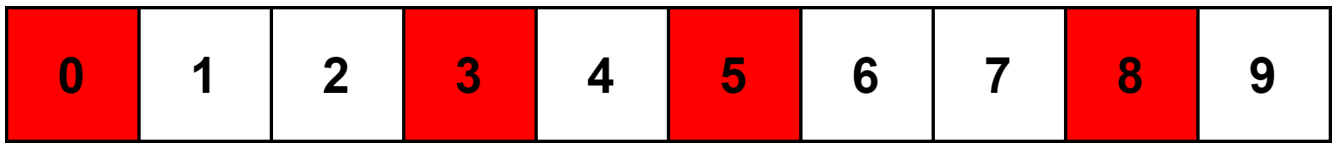
- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



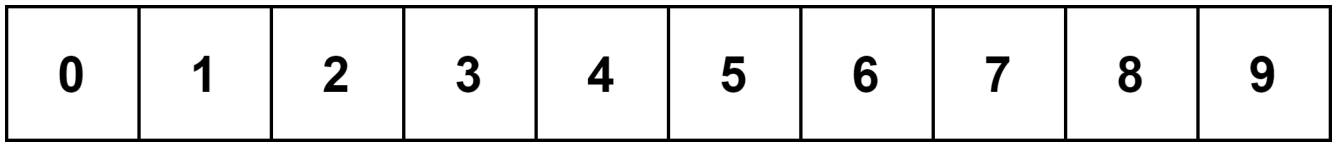
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



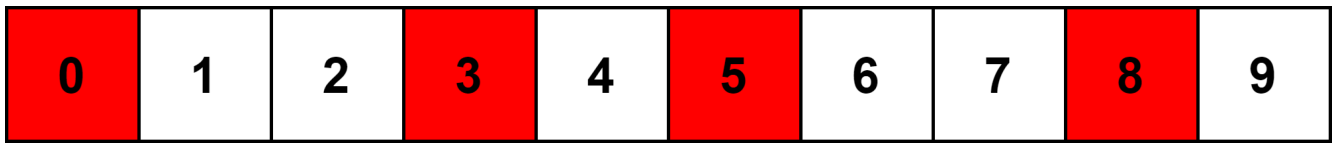
- Có thể xét các nhà kho theo thứ tự ngược lại.



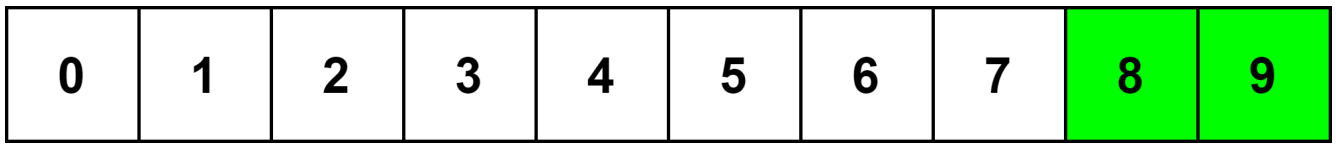
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



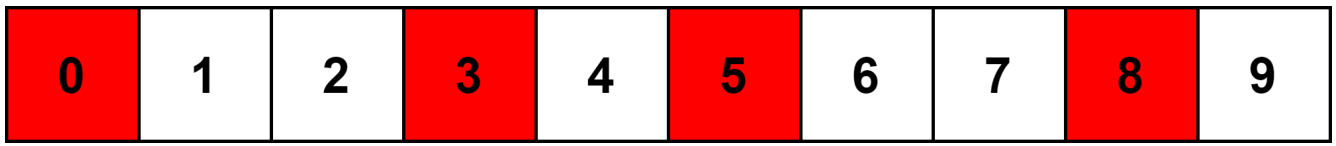
- Có thể xét các nhà kho theo thứ tự ngược lại.



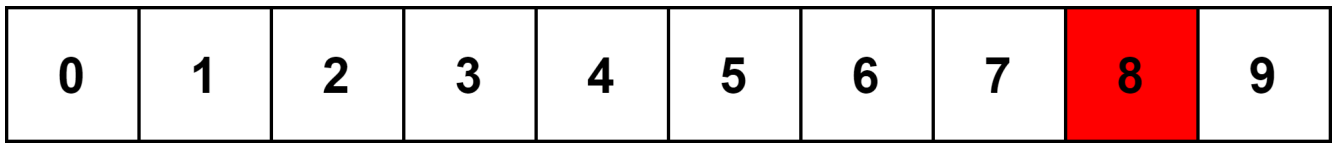
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



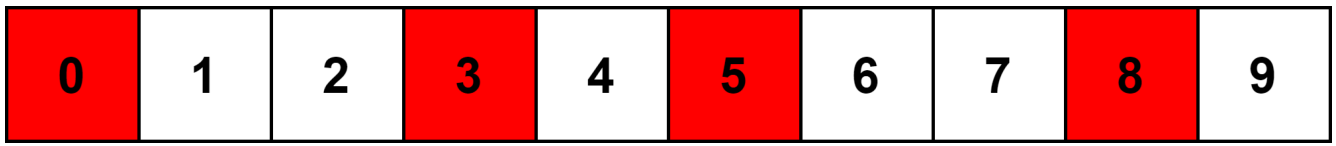
- Có thể xét các nhà kho theo thứ tự ngược lại.



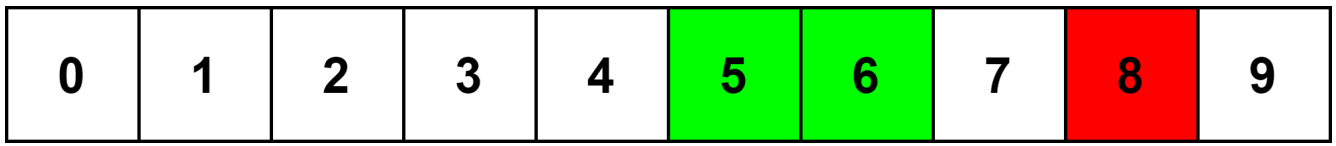
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



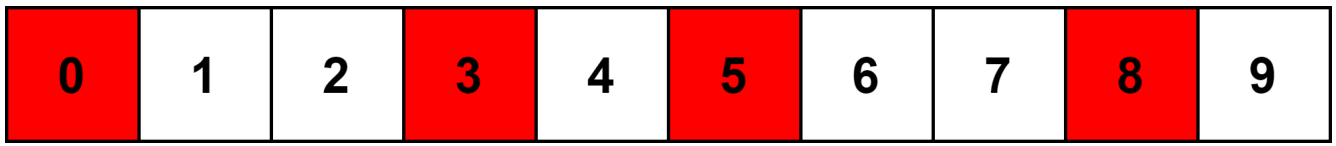
- Có thể xét các nhà kho theo thứ tự ngược lại.



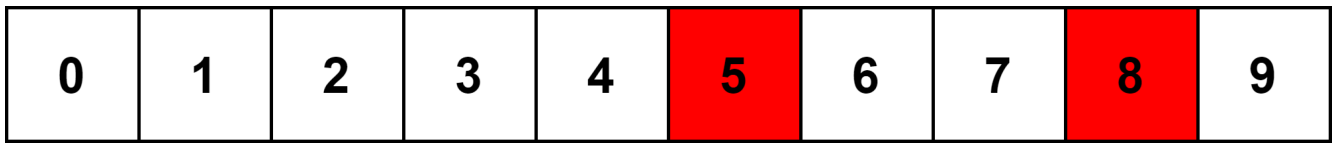
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



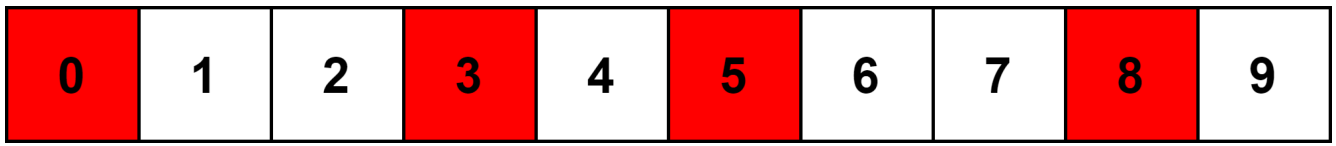
- Có thể xét các nhà kho theo thứ tự ngược lại.



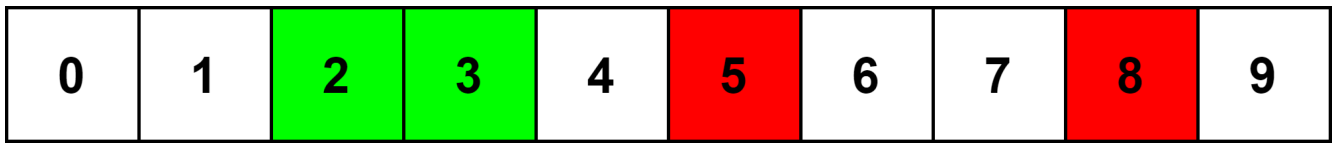
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



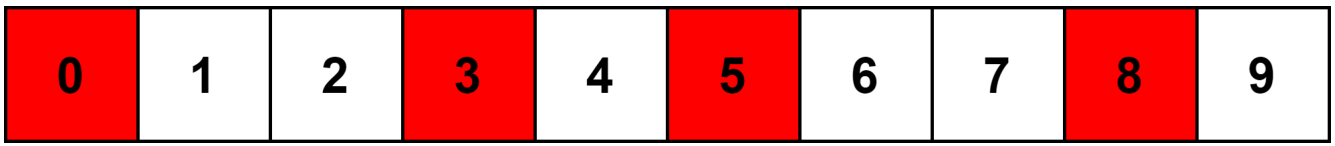
- Có thể xét các nhà kho theo thứ tự ngược lại.



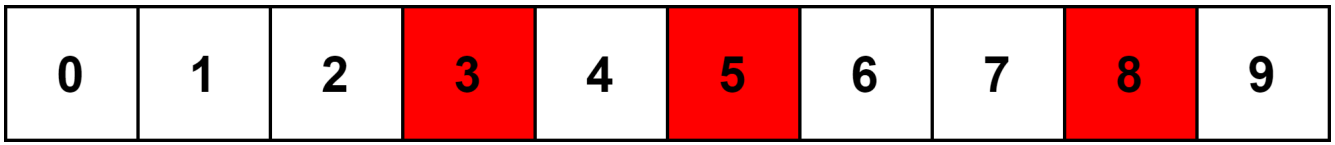
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



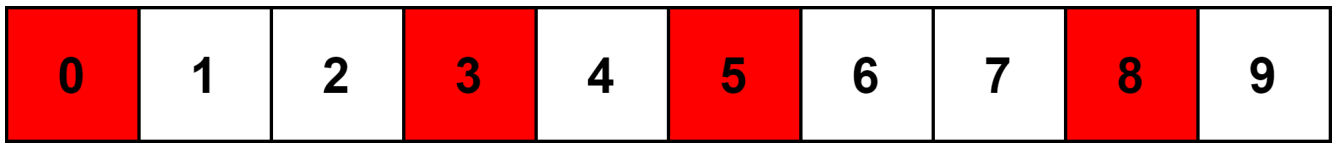
- Có thể xét các nhà kho theo thứ tự ngược lại.



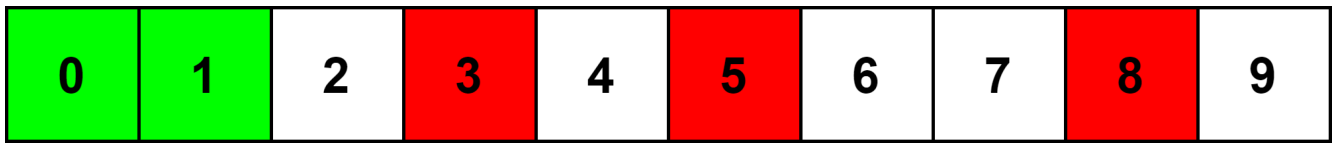
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



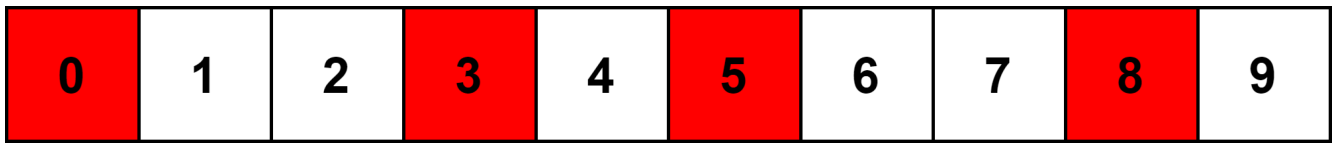
- Có thể xét các nhà kho theo thứ tự ngược lại.



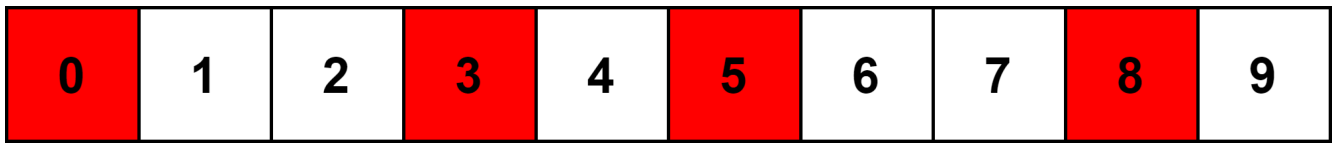
Tìm kiếm vét cạn 1b

Nhận xét:

- Nếu nhà kho thứ i được chọn, ta chỉ có thể chọn các nhà kho $[i + L_1, i + L_2] \rightarrow$ hàm đệ quy không cần gọi qua các giá trị $[i + 1, i + L_1 - 1]$.
- Ví dụ: $n = 10, L_1 = 2, L_2 = 3$:



- Có thể xét các nhà kho theo thứ tự ngược lại.



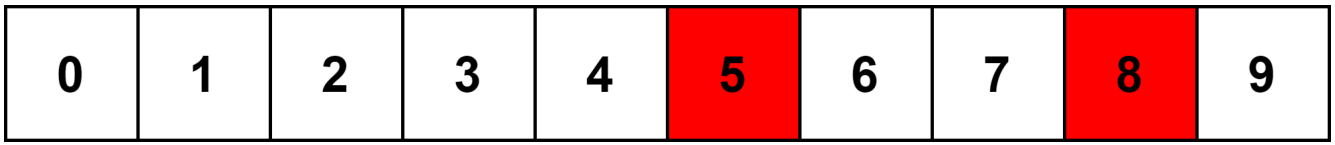
Code 1b

```
void _try(int x) {
    if (x < 0) {
        updateResult();
    }
    s.push(x);
    for (int i = x - l2; i <= x - l1; i++) {
        _try(i);
    }
    s.pop();
}

void main() {
    for (int i = n - l1; i < n; i++) {
        _try(i);
    }
}
```

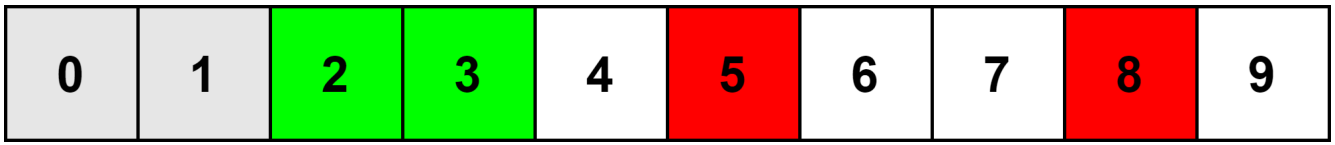
Tìm kiếm vét cạn 1c

- Sau khi chọn nhà kho x, rõ ràng ta chỉ cần quan tâm đến tổng lượng vàng lớn nhất khi chọn các nhà kho phía trước nhà kho x.



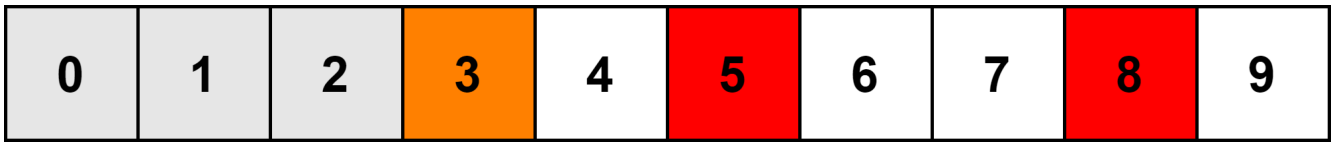
Tìm kiếm vét cạn 1c

- Sau khi chọn nhà kho x, rõ ràng ta chỉ cần quan tâm đến tổng lượng vàng lớn nhất khi chọn các nhà kho phía trước nhà kho x.



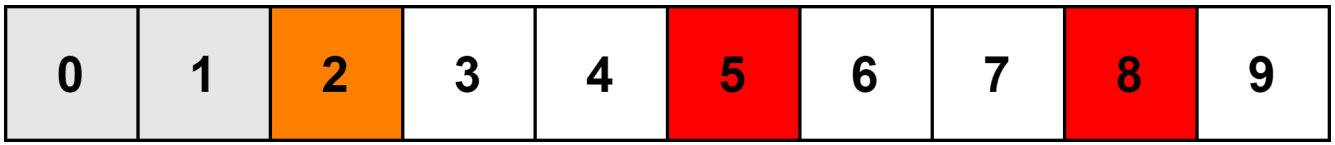
Tìm kiếm vét cạn 1c

- Sau khi chọn nhà kho x, rõ ràng ta chỉ cần quan tâm đến tổng lượng vàng lớn nhất khi chọn các nhà kho phía trước nhà kho x.

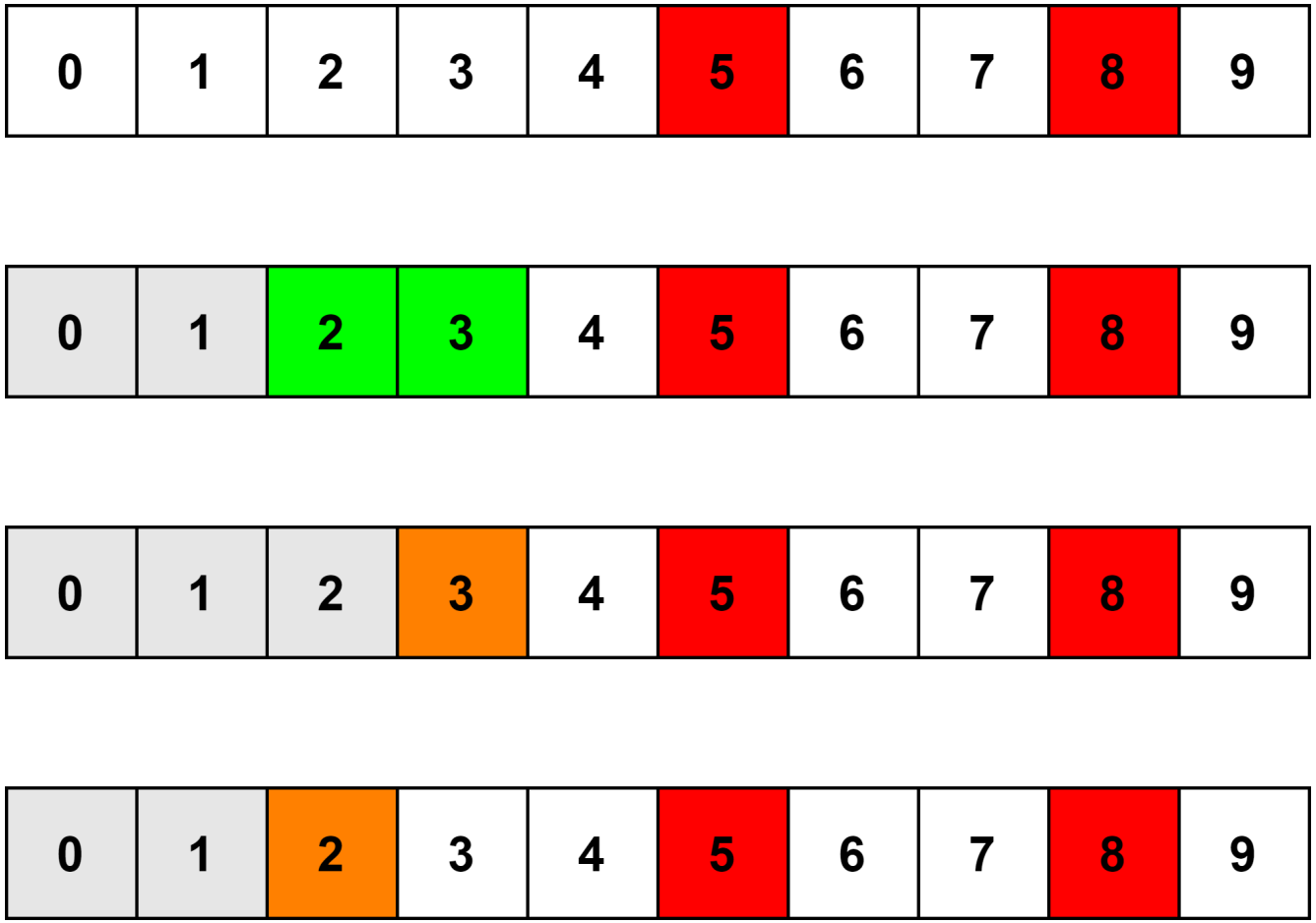


Tìm kiếm vét cạn 1c

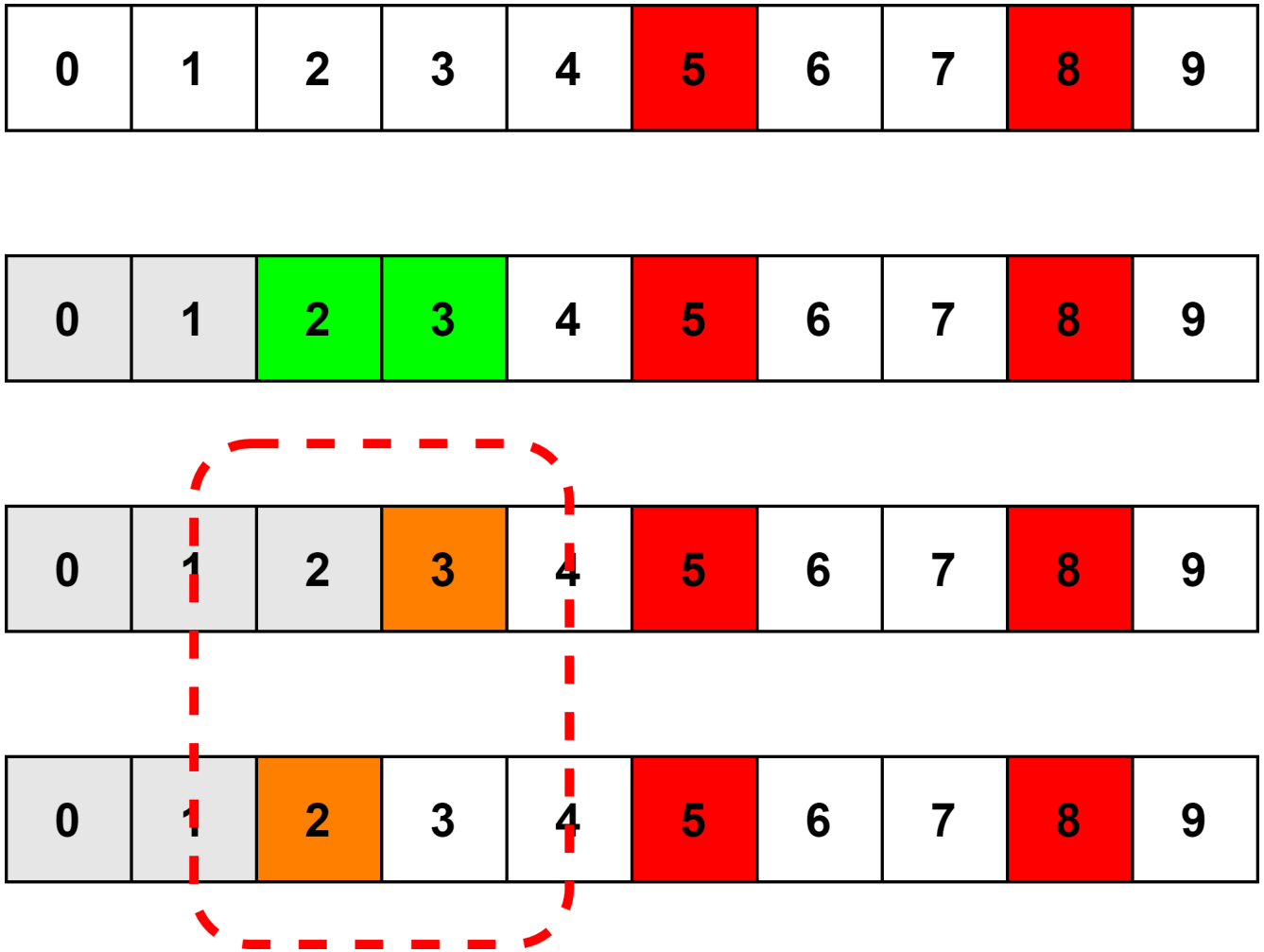
- Sau khi chọn nhà kho x, rõ ràng ta chỉ cần quan tâm đến tổng lượng vàng lớn nhất khi chọn các nhà kho phía trước nhà kho x.



Tìm kiếm vét cạn 1c



Tìm kiếm vét cạn 1c



Tìm kiếm 1c

- Sửa đổi hàm _try(x): Trả về tổng lượng vàng lớn nhất khi chọn một số nhà kho trong số các nhà kho từ 0 đến x.

Code 1c

```
int _try(int x) {
    if (x < 0) {
        return 0;
    }
    int tmp = 0;
    for (int i = x - l2; i <= x - l1; i++) {
        tmp = max(tmp, _try(i));
    }
    return tmp + a[x];
}

void main() {
    int res = 0;
    for (int i = n - l1; i < n; i++) {
        res = max(res, _try(i));
    }
}
```

Thuật toán 2a

- Công thức tính giá trị trả về của hàm _try giống công thức (1).
- Thuật toán 1c chưa tối ưu: Hàm _try được gọi nhiều lần với cùng tham số x nào đó.
- Khắc phục:
  - Lưu lại $F(x)$ là tổng lượng vàng lớn nhất khi chọn một số nhà kho trong các nhà kho từ 0 đến x.
  - Mỗi khi _try(x) được gọi, nếu $F(x)$ chưa được tính thì tính giá trị cho $F(x)$, sau đó luôn trả về $F(x)$.
- Đây chính là thuật toán quy hoạch động, sử dụng hàm đệ quy (có nhớ).

Code 2a

```
int _try(int x) {
    if (x < 0) {
        return 0;
    }
    if (F[x]) < 0) {
        int tmp = 0;
        for (int i = x - l2; i <= x - l1; i++)
            tmp = max(tmp, _try(i));
        F[x] = tmp + a[x];
    }
    return F[x];
}
void main() {
    int res = 0;
    for (int i = n - l1; i < n; i++) {
        res = max(res, _try(i));
    }
}
```

Thuật toán 2b

Ta có thể dễ dàng cài đặt thuật toán 2a bằng phương pháp lặp:

- Độ phức tạp: $O(N \times (L_2 - L_1)) = O(N^2)$.

Code 2b

```
int main() {
    ...
    for (int i = 0; i < n; i++) {
        F[i] = a[i];
    }
    for (int i = l1; i < n; i++) {
        for (int j = i - l2; j <= i - l1; j++) {
            F[i] = max(F[i], F[j] + a[i]);
        }
    }
    ...
}
```

Cải tiến thuật toán

- Nhận thấy việc tìm giá trị lớn nhất của $F[j], \forall j \in [i - L_2, i - L_1]$ khá tốn kém ($O(n)$), liệu ta có thể giảm chi phí của bước này?
- Để cải tiến thuật toán, ta cần kết hợp các cấu trúc dữ liệu nâng cao để tối ưu việc truy vấn.

Cải tiến hàm đệ quy

- Sử dụng các cấu trúc dữ liệu hỗ trợ truy vấn khoảng tốt như Segment Tree, Interval Tree (IT), Binary Index Tree (BIT).
- Các cấu trúc trên đều cho phép cập nhật một giá trị và truy vấn (tổng, min, max) trên khoảng trong thời gian $O(logn)$.
- Với bài tập này, ta cần duy trì song song 2 cấu trúc (1 để truy vấn lượng vàng lớn nhất, 1 để truy vấn các giá trị $F[x]$ chưa được tính).
- Các cấu trúc dữ liệu trên đều không được cài đặt sẵn trong thư viện và không "quá dễ hiểu".

Sử dụng hàng đợi ưu tiên

Hàng đợi ưu tiên:

- Hàng đợi ưu tiên (priority queue) là một hàng đợi có phần tử ở đầu là phần tử có độ ưu tiên cao nhất.
- Thường cài đặt bằng Heap nên có độ phức tạp cho mỗi thao tác push, pop là $O(logn)$.

Cải tiến:

- Mỗi phần tử trong hàng đợi là một cặp giá trị $(j, F[j])$.
- Ưu tiên phần tử có $F[j]$ lớn.
- Khi xét đến nhà kho i, thêm cặp giá trị $(i - L_1, F[i - L_1])$ vào hàng đợi.
- Loại bỏ phần tử j ở đầu hàng đợi trong khi $i - j > L_2$, gán $F[i] = a[i] + F[j]$.
- Độ phức tạp: $O(n + n \times log(n)) = O(n \times log(n))$

Code 3a

```
class comp {
bool reverse;
public:
    comp(const bool& revparam=false) {
        reverse=revparam;
    }

    bool operator() (const pil& lhs,
    const pil&rhs) const {
        if (reverse) {
            return (lhs.second>rhs.second);
        }
        else {
            return (lhs.second<rhs.second);
        }
    }
};
```

Code 3a

```
int main() {
    ...
    for (int i = l1; i < n; i++) {
        int j = i - l1;
        q.push(make_pair(j, f[j]));
        while (q.top().first < i - l2) {
            q.pop();
        }
        F[i] = a[i] + q.top().second;
    }
    ...
}
```

Sử dụng hàng đợi 2 đầu

Hàng đợi 2 đầu:

- Hàng đợi 2 đầu (deque) là cấu trúc dữ liệu kết hợp giữa hàng đợi và ngăn xếp $\rightarrow$ phần tử có thể được lấy ra ở đầu hoặc cuối deque.

Ta định nghĩa các thao tác push và pop cho deque dùng trong bài:

- push(x): Xoá mọi phần tử j mà $F[j] \leq F[x]$ trong hàng đợi, thêm x vào cuối hàng đợi.
- pop(): Lấy ra phần tử ở đầu hàng đợi và xoá nó khỏi hàng đợi.

Sử dụng hàng đợi 2 đầu

Áp dụng vào bài toán:

- Tính $F[i]$ theo thứ tự.
  - Gọi push(i - L1).
  - Trong khi $top() < i - L2$ gọi $pop()$.
  - $F[i] = F[top()] + a[i]$.

Sử dụng hàng đợi 2 đầu

Khi tính $F[i]$:

- Hàng đợi sắp xếp theo thứ tự giảm dần của giá trị $F[]$, do $i - L1$ được thêm vào cuối hàng đợi (khi đã loại hết các giá trị nhỏ hơn nó).
- Các nhà kho trong hàng đợi cũng được sắp xếp theo thứ tự được thêm vào hàng đợi.
- Nhà kho $i - L1$ là nhà kho cuối cùng được thêm vào hàng đợi, nên không có nhà kho nào quá gần i.
- Mọi nhà kho cách quá xa i đều bị loại khỏi hàng đợi (thao tác pop()).
- **Kết luận**: Những nhà kho còn lại trong hàng đợi đều thoả mãn ràng buộc, và nhà kho đầu tiên của hàng đợi là lựa chọn tối ưu.

Sử dụng hàng đợi 2 đầu

Độ phức tạp:

- Khi tính $F[i]$, push(i - L1) và vòng lặp các thao tác pop() đều có chi phí tối đa là $O(n)$.
- Tổng chi phí cũng chỉ là $O(n)$:
  - Mỗi nhà kho được thêm vào hàng đợi tối đa 1 lần và được lấy ra khỏi hàng đợi tối đa 1 lần.
  - n nhà kho chỉ được đưa vào và lấy ra tổng cộng $2n = O(n)$ lần.
- **Độ phức tạp**: $O(n)$.

Code 3b

```
int main() {
    ...
    for (int i = l1; i < n; i++) {
        int j = i - l1;
        dq.push(j);
        while (dq.top() < i - l2) {
            dq.pop();
        }
        F[i] = a[i] + F[dq.top()];
    }
    ...
}
```

Bàn luận

Bàn luận

Tại sao deque lại hiệu quả hơn priority queue trong trường hợp này?

Bàn luận

Tại sao deque lại hiệu quả hơn priority queue trong trường hợp này?

- Do deque luôn xoá các nhà kho chắc chắn không thể dùng đến, nên các thao tác truy vấn sau đó sẽ hiệu quả hơn.

Bàn luận

Tại sao deque lại hiệu quả hơn priority queue trong trường hợp này?

- Do deque luôn xoá các nhà kho chắc chắn không thể dùng đến, nên các thao tác truy vấn sau đó sẽ hiệu quả hơn.

Tại sao không dễ tối ưu cài đặt sử dụng hàm đệ quy?

Bàn luận

Tại sao deque lại hiệu quả hơn priority queue trong trường hợp này?

- Do deque luôn xoá các nhà kho chắc chắn không thể dùng đến, nên các thao tác truy vấn sau đó sẽ hiệu quả hơn.

Tại sao không dễ tối ưu cài đặt sử dụng hàm đệ quy?

- Do hàm đệ quy gọi _try(x) không theo thứ tự của x, nên không thể áp dụng các cấu trúc như priority queue và deque.

Bàn luận

Tại sao deque lại hiệu quả hơn priority queue trong trường hợp này?

- Do deque luôn xoá các nhà kho chắc chắn không thể dùng đến, nên các thao tác truy vấn sau đó sẽ hiệu quả hơn.

Tại sao không dễ tối ưu cài đặt sử dụng hàm đệ quy?

- Do hàm đệ quy gọi _try(x) không theo thứ tự của x, nên không thể áp dụng các cấu trúc như priority queue và deque.

Truy vết

- Hầu hết các bài toán không chỉ yêu cầu đưa ra giá trị tối ưu mà còn yêu cầu đưa ra lời giải.
- Để đưa ra lời giải, ta cần một mảng đánh dấu để có thể truy vết ngược lại. Ví dụ được thể hiện ở code bên dưới:

Code 3c

```
int main() {
    ...
    for (int i = l1; i < n; i++) {
        int j = i - l1;
        dq.push(j);
        while (dq.top() < i - l2) {
            dq.pop();
        }
        F[i] = a[i] + F[dq.top()];
        trace[i] = dq.top();
    }
    int i = argmax(F);
    while (i >= 0) {
        select.add(i);
        i = trace[i];
    }
    ...
}
```

Mục lục

1 MỞ ĐẦU

2 GOLD MINING

3 WAREHOUSE

4 RETAIL OUTLETS

5 MACHINE

6 TỔNG KẾT

KNAPSAC

- Cho N đồ vật và một cái túi có sức chứa tối đa là M. Đồ vật i có khối lượng $w[i]$ và giá trị $a[i]$.
- Tìm cách lấy một số đồ vật sao cho tổng khối lượng không vượt quá M và tổng giá trị lớn nhất có thể.

Tìm kiếm vét cạn

- Mỗi cách chọn lấy các đồ vật tương ứng với một dãy nhị phân độ dài n. Bit thứ i là 0/1 tương ứng là không lấy/có lấy đồ vật thứ i.
- Xét hết các xâu nhị phân độ dài N và tìm nghiệm tốt nhất.
- Ta có thể sử dụng vòng lặp hoặc dùng đệ quy - quay lui.
- Độ phức tạp $\mathcal{O}(2^N \times N)$.

Tìm kiếm vét cạn

Có thể cải tiến thuật toán tìm kiếm vét cạn?

Tìm kiếm vét cạn

Có thể cải tiến thuật toán tìm kiếm vét cạn?

- Slide học kỳ 2020-1.

Quy hoạch động

Quy hoạch động

- Gọi $F[i][j]$ là tổng giá trị lớn nhất nhận được khi xét i đồ vật đầu tiên và chỉ dùng một chiếc túi có sức chứa j.

Quy hoạch động

- Gọi $F[i][j]$ là tổng giá trị lớn nhất nhận được khi xét i đồ vật đầu tiên và chỉ dùng một chiếc túi có sức chứa j.
- Kết quả: $F[N][M]$.

Quy hoạch động

- Gọi $F[i][j]$ là tổng giá trị lớn nhất nhận được khi xét i đồ vật đầu tiên và chỉ dùng một chiếc túi có sức chứa j.
- Kết quả: $F[N][M]$.
- Khởi tạo:
  - $F[0][j] = 0, \forall j \geq 0$.
  - $F[i][j] = -\infty, \forall j < 0$.

Quy hoạch động

- Gọi $F[i][j]$ là tổng giá trị lớn nhất nhận được khi xét i đồ vật đầu tiên và chỉ dùng một chiếc túi có sức chứa j.
- Kết quả: $F[N][M]$.
- Khởi tạo:
  - $F[0][j] = 0, \forall j \geq 0$.
  - $F[i][j] = -\infty, \forall j < 0$.
- Công thức phụ thuộc:

$$F[i][j] = max(F[i-1][j], F[i-1][j-w[i]] + a[i]), \forall i > 0, j \geq 0 \quad (2)$$

Quy hoạch động

- Gọi $F[i][j]$ là tổng giá trị lớn nhất nhận được khi xét i đồ vật đầu tiên và chỉ dùng một chiếc túi có sức chứa j.
- Kết quả: $F[N][M]$.
- Khởi tạo:
  - $F[0][j] = 0, \forall j \geq 0$.
  - $F[i][j] = -\infty, \forall j < 0$.
- Công thức phụ thuộc:

$$F[i][j] = max(F[i-1][j], F[i-1][j-w[i]] + a[i]), \forall i > 0, j \geq 0 \quad (2)$$

- Độ phức tạp: $\mathcal{O}(N \times M)$.

So sánh độ phức tạp

- Tìm kiếm vét cạn: $\mathcal{O}(2^N \times N)$.
- Quy hoạch động: $\mathcal{O}(N \times M)$.

05. WAREHOUSE

- N nhà kho được đặt tại các vị trí $1...N$. Mỗi nhà kho có:
  - a_i là số lượng hàng.
  - t_i là thời gian lấy hàng.
- Một tuyến đường lấy hàng đi qua các trạm $x_1 < x_2 < ... < x_k$ $(1 \leq x_j \leq N, j = 1...k)$ sao cho:
  - $x_{i+1} - x_i \leq D \ \forall i \in [1, k]$.
  - $\sum_{i=1}^{k} t[x_i] \leq T$
- **Yêu cầu:** Tìm lộ trình để xe tải lấy được nhiều hàng nhất.
- Giới hạn:
  - $1 \leq N \leq 1000$.
  - $1 \leq T \leq 100$.
  - $1 \leq D, a_i, t_i \leq 10$.

Nhận xét

- Bài toán là sự kết hợp của bài GOLD MINING và bài KNAPSAC.
- $L_1 = 1, L_2 = D \leq 10 \rightarrow$ không cần sử dụng priority queue hoặc dequeue để tối ưu truy vấn.
- Yếu tố thời gian có thể xem như sức chứa của túi trong bài KNAPSAC.

Thuật toán

- Gọi $dp[i][k]$ là số lượng hàng tối đa thu được khi xét các nhà kho $1...i$, lấy hàng ở kho i và thời gian lấy hàng không quá k.
- $$dp[i][k] = \begin{cases} -\infty & \text{if } k < t[i] \\ \max(dp[j][k - t[i]] + a[i], j \in [i - D, i - 1]) & \text{if } k \geq t[i] \end{cases}$$
- kết quả $ans = max(dp[i][k], i \in [1, n], k \in [1, T])$

Code

```
for (int i = 1; i <= n; i++) {
    for (int k = t[i]; k <= T; k++) {
        for (int j = i-1; j >= max(0,i-D); j--)
            dp[i][k] = max(dp[i][k],
                dp[j][k-t[i]] + a[i]);
        ans = max(ans, dp[i][k]);
    }
}
```

Bài toán tương tự

- DRONE PICKUP: Slide học kỳ 2020-1.

Mục lục

1 MỞ ĐẦU

2 GOLD MINING

3 WAREHOUSE

4 RETAIL OUTLETS

5 MACHINE

6 TỔNG KẾT

05. RETAIL OUTLETS

- Cần phân bổ M cửa hàng cho N chi nhánh.
- Số cửa hàng được phân bổ cho chi nhánh i phải dương và chia hết cho số nhân viên bán hàng $a[i]$ của chi nhánh này.
- Hai cách phân bổ được gọi là khác nhau nếu số có ít nhất một chi nhánh có số cửa hàng được phân bổ khác nhau.
- **Yêu cầu:** Đếm số cách phân bổ khác nhau.
- Giới hạn:
  - $1 \leq N \leq 100, 1 \leq M \leq 500$.
  - $1 \leq a_i \leq 10$.

Thuật toán

- Gọi $F[i][j]$ là số cách phân bổ j cửa hàng cho i chi nhánh đầu tiên.
- Kết quả: $F[N][M]$.
- Khởi tạo: $F[0][0] = 1$.
- Công thức phụ thuộc:

$$F[i][j] = \sum_{k>0, k \vdots a[i]} F[i-1][j-k] \tag{3}$$

Code

```
f[0][0] = 1;
for (int i = 1; i <= n; i++) {
    for (int j = a[i]; j <= m; j++) {
        for (int k = a[i]; k <= j; k += a[i])
            f[i][j] += f[i-1][j-k]
                %= 1000000007;
    }
}
```

Mục lục

1 MỞ ĐẦU

2 GOLD MINING

3 WAREHOUSE

4 RETAIL OUTLETS

5 MACHINE

6 TỔNG KẾT

05. MACHINE

- Cho N đoạn, đoạn i bắt đầu từ s_i và kết thúc tại t_i, có giá trị $v_i = t_i - s_i$.
- **Yêu cầu:** Chọn 2 đoạn i, j sao cho $t_i < s_j$ hoặc $t_j < s_i$ có tổng giá trị lớn nhất.
- Giới hạn:
  - $2 \leq n \leq 2 \times 10^6$.
  - $1 \leq s_i < t_i \leq 2 \times 10^6$.

Thuật toán 1

- Duyệt toàn bộ $\frac{n(n-1)}{2}$ lựa chọn, với mỗi lựa chọn kiểm tra ràng buộc và cập nhật lời giải.
- Độ phức tạp: $\mathcal{O}(n^2)$.

Thuật toán 2

- Nếu chọn j làm đoạn sau, đoạn i được chọn cùng phải thỏa mãn $t_i < s_j$. Ta muốn chọn đoạn i có giá trị lớn nhất thỏa mãn điều kiện này.
- Gọi $maxV[x]$ là giá trị của đoạn tốt có thời điểm kết thúc $\leq x$.
- Kết quả: $max_j(v_j + maxV[s_j - 1])$.

Thuật toán 2

Cách tính $maxV[x]$:

- Gọi $\_maxV[x]$ là giá trị đoạn tốt nhất kết thúc tại đúng thời điểm x:
  - Khởi tạo $\_maxV[x] = 0, \forall x$.
  - Với mỗi đoạn i, cập nhật $\_maxV[t_i] = max(\_maxV[t_i], v_i)$.
- $maxV[x] = max(maxV[x-1], \_maxV[x])$.

Thuật toán 2

Độ phức tạp:

- Tìm $\_maxV[x]$: $\mathcal{O}(n)$.
- Tìm $maxV[x]$: $\mathcal{O}(M) = \mathcal{O}(N)$, $M = max_i(t_i) - min_i(s_i)$.
- Tìm $max_j(v_j + maxV[s_j - 1])$: $\mathcal{O}(n)$.
- Tổng: $\mathcal{O}(n)$.

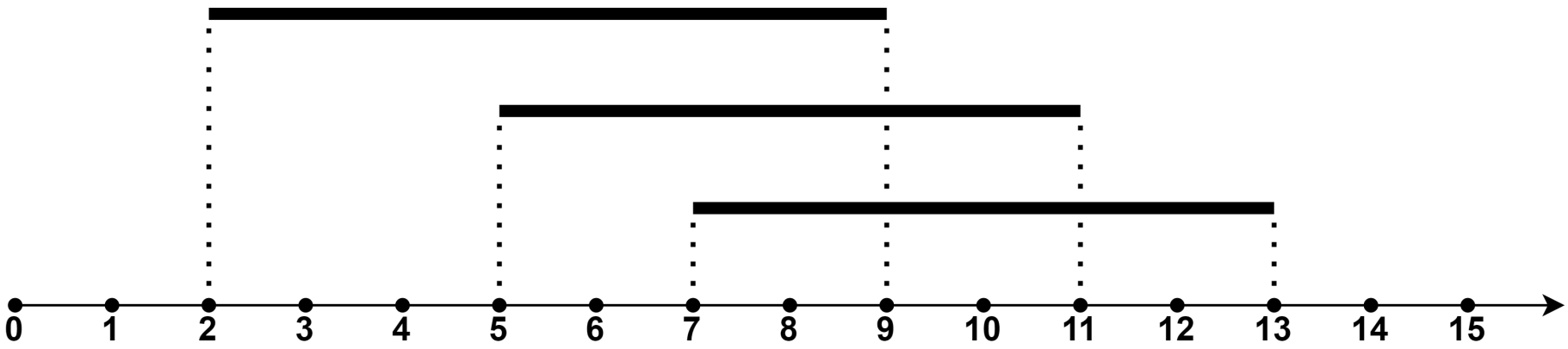
Code

```
int main() {
  const int M = 2e6 + 5;
  for (int i = 1; i <= n; i+)
    maxV[t[i]] = max(maxV[t[i]], v[i]);
  for (int i = 1; i < M; i++)
    maxV[i] = max(maxV[i-1], maxV[i]);
  long long ans = -1;
  for (int j = 1; j <= n; j++)
    if (maxV[s[i] - 1] > 0
      ans = max(ans, maxV[s[i] - 1] + v[i]);
  cout << ans << endl;
}
```

Nhận xét

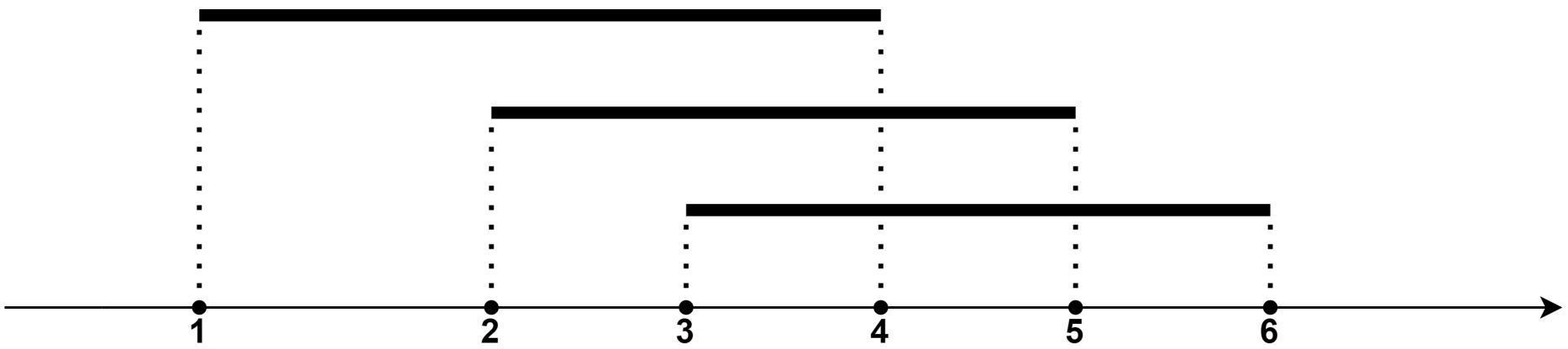
- Nếu s_i và t_i không có giới hạn, độ phức tạp là $\mathcal{O}(M)$.

Rời rạc hóa



Hình 1: Các mốc thời gian ban đầu

Rời rạc hóa



Hình 2: Các mốc thời gian rời rạc hóa

Mục lục

1. MỞ ĐẦU
2. GOLD MINING
3. WAREHOUSE
4. RETAIL OUTLETS
5. MACHINE
6. TỔNG KẾT

TỔNG KẾT

Các bước giải bài toán bằng Quy hoạch động:

TỔNG KẾT

Các bước giải bài toán bằng Quy hoạch động:

- Xác định bài toán con:
  - Xác định các chiều của bài toán con: items, resources,...
  - Định nghĩa bài toán con dựa trên các chiều đã xác định.

TỔNG KẾT

Các bước giải bài toán bằng Quy hoạch động:

- Xác định bài toán con:
  - Xác định các chiều của bài toán con: items, resources,...
  - Định nghĩa bài toán con dựa trên các chiều đã xác định.
- Tìm trường hợp cơ sở: những trường hợp dễ dàng biết kết quả.

TỔNG KẾT

Các bước giải bài toán bằng Quy hoạch động:

- Xác định bài toán con:
  - Xác định các chiều của bài toán con: items, resources,...
  - Định nghĩa bài toán con dựa trên các chiều đã xác định.
- Tìm trường hợp cơ sở: những trường hợp dễ dàng biết kết quả.
- Tìm công thức phụ thuộc vào lời giải của bài toán con.
  - Xác định các lựa chọn có thể.
  - Với mỗi lựa chọn, xác định công thức phụ thuộc.
  - Tổng hợp các lựa chọn.

TỔNG KẾT

Các bước giải bài toán bằng Quy hoạch động:

- Xác định bài toán con:
  - Xác định các chiều của bài toán con: items, resources,...
  - Định nghĩa bài toán con dựa trên các chiều đã xác định.
- Tìm trường hợp cơ sở: những trường hợp dễ dàng biết kết quả.
- Tìm công thức phụ thuộc vào lời giải của bài toán con.
  - Xác định các lựa chọn có thể.
  - Với mỗi lựa chọn, xác định công thức phụ thuộc.
  - Tổng hợp các lựa chọn.
- Truy vết:
  - Lưu vết khi tổng hợp các lựa chọn.
  - Truy vết bằng cách đi ngược vết đã lưu, xuất phát từ lời giải tối ưu, kết thúc khi chạm trường hợp cơ sở.

Thuật toán ứng dụng
Bài thực hành số 4: Quy hoạch động

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương

Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 16 tháng 5 năm 2021