綜合技巧

日月卦長

include 大多數 STL (gcc only)

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    sort(a.begin(), a.end());
    for (int i = 0; i < n; i++) {
        cout << a[i] << " \n"[i == n - 1];
    }
    return 0;
}
```

I/O 優化

- 高中的題目有時候會出現這樣的敘述
  「建議使用 scanf / printf，使用 cin 和 cout 可能造成 TLE。」

- 但這句話其實不太正確
  事實上 cin, cout 因為某些原因速度被調慢了

- 我們可以手動把變慢的因素拿掉

std::cin.tie(nullptr);

在文字被輸出前，它會先被送到一個「緩衝區」
等「緩衝區」中文字夠多的時候才會被真正輸出

cin 認為要求使用者輸入之前，要先讓使用者看見先前輸出的文字
所以執行 cin 前預設會先強制釋放(輸出)「緩衝區」的內容

執行 std::cin.tie(nullptr); 後可以取消強制釋放的行為

不要用 endl
改用 '\n'

有個強制釋放「緩衝區」的指令：
std::cout << std:: flush;

<< std::endl
等同於
<< std:: flush << '\n'

std::ios::sync_with_stdio(false);

C++ 中有兩種輸入輸出系統：stdio 和 iostream

為了讓 iostream 和 stdio 可以混合使用
C++ 有對 iostream 做一些額外的操作讓速度變慢

通常寫題目我們只會選擇其中一種系統
因此可以呼叫 std::ios::sync_with_stdio(false);
關閉混合使用的功能

I/O 優化精簡寫法

```cpp
#include <bits/stdc++.h>
using namespace std;
bool IO_opt = cin.tie(0)->sync_with_stdio(0);

int main() {
  // you code here
  return 0;
}
```

我把註解註解了

```
/*
int a;
int b;
// */
```

```
// /*
int a;
int b;
// */
```

我把註解註解了

```
// /*
int a;
int b;
/*/
int c;
int d;
// */
```

```
/*
int a;
int b;
/*/
int c;
int d;
// */
```

趨近於 0

```
while( x --> 0 )
{
  printf("%d ", x);
}
```

請問 C++ 的這個 "-->" 運算子怎麼念?

靠北工程師

你應該要這麼理解才對

```
while((x--) > 0) {
    cout << x << endl;
}
```

Functor

```
struct functor {
    int operator() (int a) {
        return a * a;
    }
};

functor func;
cout << func(5) << '\n';
```

Functor and std::function

```
#include <functional>

struct functor {
    bool operator() (int a, int b) const {
        return a > b;
    }
};
int main() {
    std::function<bool(int, int)> cmp{functor()};
    vector<int> v = {4, 2, 5, 1, 3};
    sort(v.begin(), v.end(), cmp);
    return 0;
}
```

跟我說說錯在哪邊

```
struct A {
  map<int, int> M;
  void for_each(function<void(const pair<int, int> &)> callback) const {
    for (const auto &P : M) {
      callback(P);
    }
  }
} a;
for (int i = 0; i < 10; ++i)
  a.M[i] = 0;
a.for_each(
    [](const auto &p) { const_cast<pair<int, int> &>(p).second = 7122; });
for (int i = 0; i < 10; ++i)
  cout << a.M[i] << endl;
```

Functor and Lambda [&](){}

CppInsights

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  int x = 0;
  auto get_and_increase_x = [&]() {
    x += 1;
    return x;
  };
  cout << get_and_increase_x() << endl;
  cout << x << endl;
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  int x = 0;
  struct Lambda {
    int &x;
    Lambda(int &x) : x(x) {}
    int operator()() {
      x += 1;
      return x;
    }
  };
  auto get_and_increase_x = Lambda(x);
  cout << get_and_increase_x() << endl;
  cout << x << endl;
  return 0;
}
```

Functor and Lambda [=](){}

CppInsights

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  int x = 0;
  auto get_x = [=]() {
    return x;
  };
  ++x;
  cout << get_x() << endl;
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  int x = 0;
  struct Lambda {
    int operator()() const { return x; }
    int x;
    Lambda(int &x) : x(x) {}
  };
  auto get_x = Lambda(x);
  ++x;
  cout << get_x() << endl;
  return 0;
}
```

Functor and Lambda [](){}

CppInsights

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  vector<int> vec = {5, 3, 8, 1, 2};
  sort(vec.begin(), vec.end(),
    [](auto a, auto b) {
      return a > b;
    });
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Lambda {
  template <typename T, typename U>
  auto operator()(T a, U b) { return a < b; }
};
int main() {
  vector<int> vec = {5, 3, 8, 1, 2};
  sort(vec.begin(), vec.end(), Lambda());
  return 0;
}
```

Functor and Lambda

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  vector<int> v(5);
  int x = 10, y = 0;
  auto func = [&, x, &z = y]() {
    for (auto &val : v) {
      val = x;
      z += x;
    }
  };
  func();
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  vector<int> v(5);
  int x = 10, y = 0;
  struct Lambda {
    int x;
    int &z;
    vector<int> &v;
    void operator()() {
      for (auto &val : v) {
        val = x;
        z += x;
      }
    }
    Lambda(int x, int &z, vector<int> &v)
      : x(x), z(z), v(v) {}
  };
  auto func = Lambda(x, y, v);
  func();
  return 0;
}
```

structured binding 在 C++20 才能被 capture

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  auto [a, b] = make_pair(1, 2);
  auto func = [&]() { return a + b; };
  cout << func() << endl;
  return 0;
}
```

Only work after C++20

Work in C++17

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  auto [a, b] = make_pair(1, 2);
  auto func = [&a = a, &b = b]() {
    return a + b;
  };
  cout << func() << endl;
  return 0;
}
```

沒有參數時可以省略 ()

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  const auto max_val = []{
    int a = 10;
    int b = 20;
    return max(a, b);
  }();
  return 0;
}
```

Lambda 指定回傳型態

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  char *env = std::getenv("MANE");
  string name = [&] {
    if (env) {
      return env;
    }
    return string();
  }();
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  char *env = std::getenv("MANE");
  string name = [&]() -> string {
    if (env) {
      return env;
    }
    return string();
  }();
  return 0;
}
```

Lambda 與遞迴

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  std::function<int(int n, int m)> C = [&](int n, int m) {
    if (m == 0 || m == n)
      return 1;
    return C(n - 1, m - 1) + C(n - 1, m);
  };
  cout << C(10, 5) << endl;
  return 0;
}
```

Lambda 與遞迴

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  map<int, int *> mp1;
  map<int, map<int, int *>> mp2;
  map<int, map<int, map<int, int *>>> mp3;
  auto cleanup = [](auto &&self, auto &m) -> void {
    for (auto &p : m) {
      if constexpr (is_pointer_v<decltype(p.second)>) {
        delete p.second;
      } else {
        self(self, p.second);
      }
    }
    m.clear();
  };
  cleanup(cleanup, mp1);
  cleanup(cleanup, mp2);
  cleanup(cleanup, mp3);
  return 0;
}
```

template argument deduction C++17

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  vector<vector<int>> v(10, vector<int>(10, 0));
  auto v2 = vector(10, vector(10, 0));
  return 0;
}
```

徹底刪除 STL Container 的記憶體

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  map<int, string> A;
  map<int, map<int, long long>> B;
  vector<map<int, string>> C;
  auto release_memory = [](auto &m) {
    decay_t<decltype(m)>().swap(m);
  };
  release_memory(A);
  release_memory(B);
  release_memory(C);
  return 0;
}
```

vector<bool> 空間優化

- 在 C++ 標準庫中做了特殊空間優化：
  - 它不是用一個 bool 佔一個 byte，而是用一個 bit 來存一個 bool
- 假設有個 vector<bool> V，底下是一些編譯會爛掉的操作
  - V[100] &= true
  - auto &ref = V[100]
  - ++V[100]
- 一般來說競賽使用是沒有任何問題的

```cpp
#include <bits/stdc++.h>
int main() {
  for (int _ = 1000; --_;) {
    std::vector<bool> V(64, false);
    std::vector<std::thread> threads;
    constexpr int num_threads = 8;
    for (int t = 0; t < num_threads; ++t) {
      threads.emplace_back([&, t]() {
        for (int i = t; i < 64; i += num_threads) {
          V[i] = true;
          std::this_thread::sleep_for(std::chrono::nanoseconds(1));
        }
      });
    }
    for (auto &th : threads) th.join();
    if (count(V.begin(), V.end(), 0) != 0) {
      for (auto v : V) std::cout << v;
      std::cout << std::endl;
      break;
    }
  }
  return 0;
}
```

但平行寫入會出問題

猜猜誰比較快

```cpp
// suppose 0 <= lhs, rhs < mod
int mod_add_0(int lhs, int rhs, int mod) {
  return (lhs + rhs) % mod;
}
int mod_sub_0(int lhs, int rhs, int mod) {
  return ((lhs - rhs) % mod + mod) % mod;
}
```

```cpp
// suppose 0 <= lhs, rhs < mod
int mod_add_2(int lhs, int rhs, int mod) {
  lhs += rhs;
  return (lhs >= mod ? lhs - mod : lhs);
}
int mod_sub_2(int lhs, int rhs, int mod) {
  lhs -= rhs;
  return (lhs < 0 ? lhs + mod : lhs);
}
```

```cpp
// suppose 0 <= lhs, rhs < mod
constexpr int mod = 1e9 + 7;
int mod_add_1(int lhs, int rhs) {
  return (lhs + rhs) % mod;
}
int mod_sub_1(int lhs, int rhs) {
  return ((lhs - rhs) % mod + mod) % mod;
}
```

```cpp
// suppose 0 <= lhs, rhs < mod
constexpr int mod = 1e9 + 7;
int mod_add_3(int lhs, int rhs) {
  lhs += rhs;
  return (lhs >= mod ? lhs - mod : lhs);
}
int mod_sub_3(int lhs, int rhs) {
  lhs -= rhs;
  return (lhs < 0 ? lhs + mod : lhs);
}
```

誰比較快

```cpp
void benchmark(const string &name, function<int(int, int)> func) {
  int lhs = 123456789, rhs = 987654321;
  auto start = high_resolution_clock::now();
  for (long long i = 0; i < N; ++i) func(lhs, rhs);
  auto end = high_resolution_clock::now();
  cout << name << ": " << duration_cast<milliseconds>(end - start).count()
       << " ms\n";
}
int main() {
  int MD = 1e9 + 7; // 防止編譯器優化
  benchmark("add_0", [&](int a, int b) { return mod_add_0(a, b, MD); });
  benchmark("add_1", [](int a, int b) { return mod_add_1(a, b); });
  benchmark("add_2", [&](int a, int b) { return mod_add_2(a, b, MD); });
  benchmark("add_3", [](int a, int b) { return mod_add_3(a, b); });
  benchmark("sub_0", [&](int a, int b) { return mod_sub_0(a, b, MD); });
  benchmark("sub_1", [](int a, int b) { return mod_sub_1(a, b); });
  benchmark("sub_2", [&](int a, int b) { return mod_sub_2(a, b, MD); });
  benchmark("sub_3", [](int a, int b) { return mod_sub_3(a, b); });
  return 0;
}
```

誰比較快

-O2

```
add_0: 1246 ms
add_1: 863 ms
add_2: 870 ms
add_3: 861 ms
sub_0: 4315 ms
sub_1: 1176 ms
sub_2: 1041 ms
sub_3: 856 ms
```

-Ofast

```
add_0: 1232 ms
add_1: 856 ms
add_2: 856 ms
add_3: 831 ms
sub_0: 2432 ms
sub_1: 1143 ms
sub_2: 1035 ms
sub_3: 854 ms
```

CSES 1745 - Money Sums

- 給你 $N(N \leq 100)$ 個硬幣，第 i 個硬幣面額為 $x_i(x_i \leq 1000)$
- 請輸出這些硬幣能夠組成的所有面額

CSES 1745 - Money Sums

Input

```
4
4 2 5 2
```

Output

```
9
2 4 5 6 7 8 9 11 13
```

CSES 1745 - Money Sums

- 設 $dp_{n,k} = true \leftrightarrow$ 前 n 個硬幣，面額 k 可以被創造出來
- 狀態轉移式

$$\begin{cases} dp_{0,0} = true \\ dp_{n,k} = dp_{n-1,k} \mid dp_{n-1,k-x_n} \end{cases}$$

面額 k 本來就可被前 $n-1$ 個硬幣湊出

面額 $k - x_n$ 可被前 $n-1$ 個硬幣湊出再補一個 n 號硬幣湊出面額 k

CSES 1745 - Money Sums

- 設 $dp_{n,k} = true \leftrightarrow$ 前 n 個硬幣，面額 k 可以被創造出來
- 狀態轉移式

$$\begin{cases} dp_{0,0} = true \\ dp_{n,k} = dp_{n-1,k} \mid dp_{n-1,k-x_n} \end{cases}$$

- 狀態數 $O(NK), K = \sum x_i$，狀態轉移 $O(1)$
- 時間複雜度 $O(NK)$
  - 但是 $N \leq 10^2, K \leq N \times 10^3 \leq 10^5$
  - $N \times K \leq 10^7$ 可以接受但有點大

CSES 1745 - Money Sums

```cpp
#include <bits/stdc++.h>
using namespace std;
void solve(const vector<int> &x) {
  int N = x.size();
  int K = accumulate(x.begin(), x.end(), 0);
  auto dp = vector(N + 1, vector<uint8_t>(K + 1, false));
  dp[0][0] = true;
  for (int n = 1; n <= N; ++n) {
    for (int k = 0; k <= K; ++k) {
      dp[n][k] = dp[n - 1][k];
      if (k >= x[n - 1])
        dp[n][k] |= dp[n - 1][k - x[n - 1]];
    }
  }
  // dp[n][k] == true 代表金額 k 可被湊出
}
```

狀態壓縮

```cpp
void solve(const vector<int> &x) {
  int N = x.size();
  int K = accumulate(x.begin(), x.end(), 0);
  vector<uint8_t> dp(K + 1, false);
  dp[0] = true;
  for (int i = 0; i < N; ++i) {
    for (int k = K; k >= x[i]; --k) {
      dp[k] |= dp[k - x[i]];
    }
  }
  // dp[k] == true 代表金額 k 可被湊出
}
```

bitset 加速 (32 or 64 個 bit 一起算)

```cpp
void solve(const vector<int> &x) {
  int N = x.size();
  int K = accumulate(x.begin(), x.end(), 0);
  bitset<100001> dp; // K <= 100000
  dp[0] = 1;
  for (int i = 0; i < N; ++i) {
    dp |= (dp << x[i]);
  }
  // dp[k] == 1 表示金額 k 可被湊出
}
```

時間複雜度 $O\left(\frac{NK}{B}\right), B = 32,64$ 視系統而定

Prefix Sum 前綴和

- 給定一個序列 $A = (a_1, a_2, \ldots, a_n)$
- 有 m 筆詢問，詢問包含 L, R ，請輸出
$$sum(A[L, R]) = a_L + a_{L+1} + \ldots + a_R$$
- $1 \leq n, m \leq 10^6$

Prefix Sum 前綴和

- 高中數學：

- 套用到程式：
  - $O(n)$ 存 Prefix Sum
  - $O(1)$ 算 $sum(A[L,R])$
  - 時間複雜度：$O(n+m)$

```
vector<int> S(n + 1, 0);
for (int i = 1; i <= n; i++) {
  S[i] = S[i - 1] + A[i];
}

auto sum = [&](int L, int R) {
  return S[R] - S[L - 1];
};
```

CSES 1661 - Subarray Sums II

- 給你長度是 $n(n \leq 2 \cdot 10^5)$ 的 int 陣列 A
  以及整數 $x(-10^9 \leq x \leq 10^9)$

- 請你計算 A 中所有區間總合為 x 的區間有幾個

- 顯然直接枚舉區間和很慢
  就算用前綴和加速枚舉複雜度都還是 $O(n^2)$

CSES 1661 - Subarray Sums II

Input

```
5 7
2 -1 3 5 -2
```

Output

```
2
```

CSES 1661 - Subarray Sums II

- 我們已經知道

$$sum(A[L,R]) = S_R - S_{L-1}$$

- 題目可以轉變為有多少組 (L,R) 使得 $S_R - S_{L-1} = x$
- 我們可以枚舉 R，則符合條件的 L 必須滿足

$$S_{L-1} = S_R - x$$

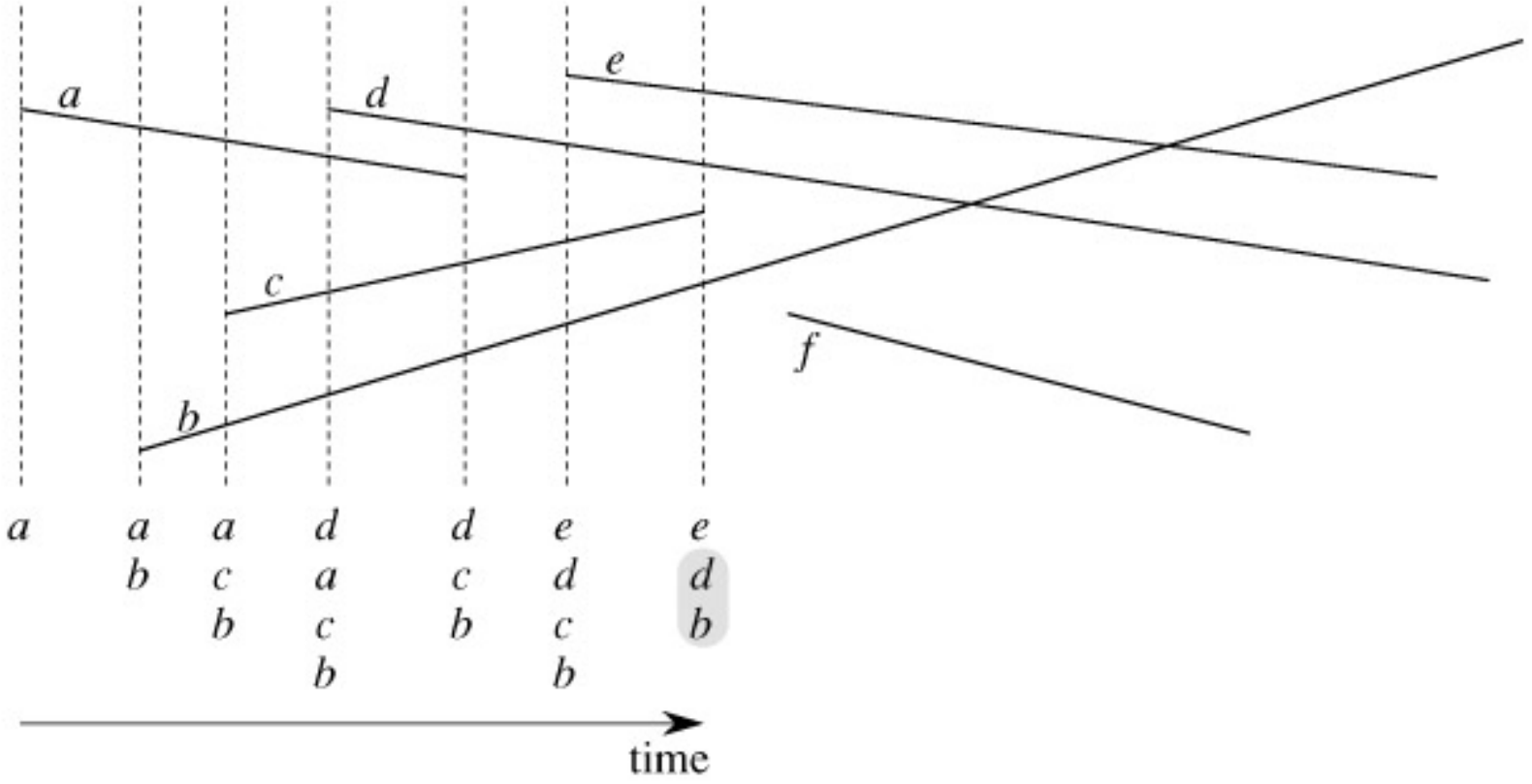
- 用 map 就可以記錄某個值出現了幾次

CSES 1661 - Subarray Sums II $O(n \log n)$

```cpp
int main() {
  int n, x;
  cin >> n >> x;
  vector<int> A(n + 1), S(n + 1, 0);
  for (int i = 1; i <= n; ++i) {
    cin >> A[i];
    S[i] = S[i - 1] + A[i];
  }
  map<long long, long long> MP;
  MP[0] = 1;
  long long ans = 0;
  for (int R = 1; R <= n; ++R) {
    ans += MP[S[R] - x];
    ++MP[S[R]];
  }
  cout << ans << "\n";
  return 0;
}
```

Sweeping Line 掃描線概念

- 二維的資料 (時間軸 / 坐標軸)，用其中一個維度排序
- 按排序順序走訪資料，利用某種資料結構維護另一個維度的資訊



CSES 1619 - Restaurant Customers

- 今天有 n 個人到達與離開餐廳的時間
  請問在任意時間點中，餐廳的最多人數是多少？

- 第一行有一個正整數 $n(n \leq 2 \cdot 10^5)$
  接下來有 n 行，每行有兩個相異正整數 $a, b(a, b \leq 10^9)$，分別代表第 i 個顧客到達與離開餐廳的時間

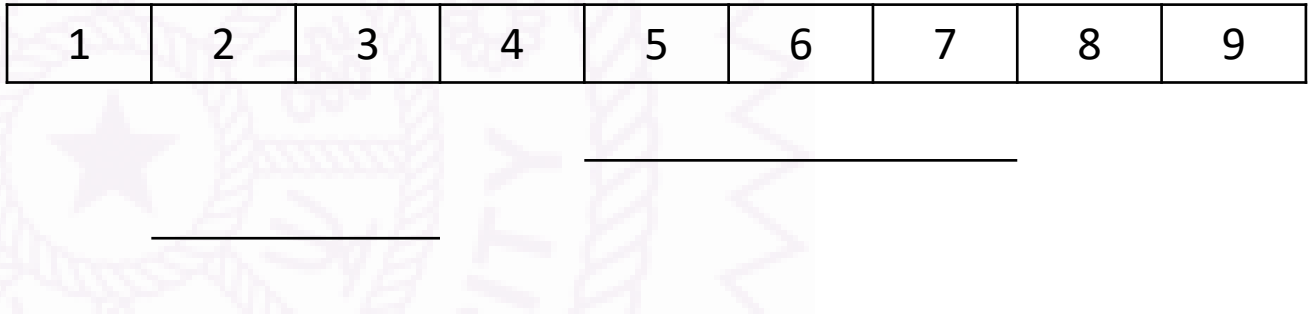
範例測資

Input

```
3
5 8
2 4
3 9
```

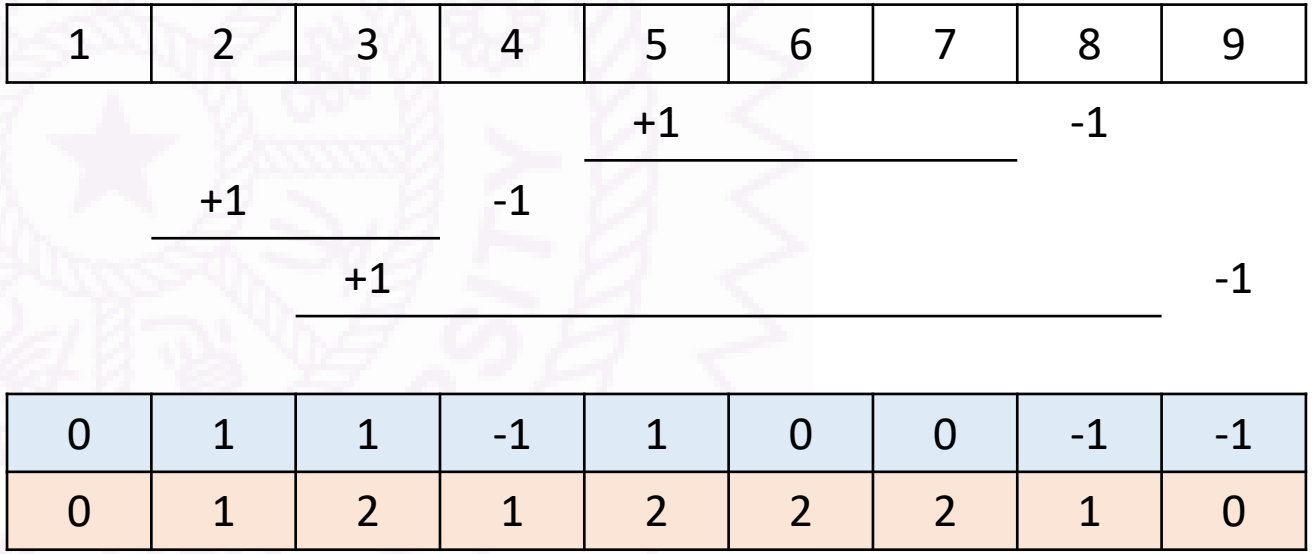
Output

```
2
```



想法

- 建立時間軸表格
- 有人到達就把該時間 +1
- 有人離開就把該時間 -1
- 用前綴和維護當前人數



好像哪裡怪怪的

這不會炸嗎?

```cpp
int main() {
  input();
  int N = 1e9 + 1;
  vector<int> timeline(N, 0);
  for (auto [a, b] : p) {
    ++timeline[a];
    --timeline[b];
  }
  int ans = 0, pre = 0;
  for (int i = 0; i < N; ++i) {
    pre += timeline[i];
    ans = max(ans, timeline[i]);
  }
  cout << ans << endl;
  return 0;
}
```

```cpp
vector<pair<int, int>> p;
void input() {
  int n;
  cin >> n;
  p.resize(n);
  for (auto &[a, b]: p) {
    cin >> a >> b;
  }
}
```

Relabel 離散化

- 給定一個數列
  - 數值大小不重要
  - 元素之間大小關係重要

$$A = (4,8,7,6,6,6,3) \rightarrow A' = (1,4,3,2,2,2,0)$$

- 想法
  - 將序列中的數值改為在序列中的名次
  - 縮小值域
    - $n \leq 10^6, a_i \leq 10^9 \rightarrow 0 \leq a_i < n$

方法一：set + map 硬幹，但很慢

```cpp
template <typename T>
auto relabel(const vector<T> &arr) {
  set<T> s(arr.begin(), arr.end());
  map<T, int> mp;
  int idx = 0;
  for (auto x : s) mp[x] = idx++;
  vector<int> ans;
  for (auto &x : arr) ans.emplace_back(mp[x]);
  return ans;
}
```

方法二：binary search

```cpp
template <typename T>
auto relabel(const vector<T> &arr) {
  auto tmp = arr;
  sort(tmp.begin(), tmp.end());
  tmp.erase(unique(tmp.begin(), tmp.end()), tmp.end());
  vector<int> ans;
  for (auto x : arr) {
    ans.emplace_back(lower_bound(tmp.begin(), tmp.end(), x) - tmp.begin());
  }
  return ans;
}
```

拆成兩個函數方便使用

```cpp
template <typename T>
auto get_unique_sorted(vector<T> arr) {
  sort(arr.begin(), arr.end());
  arr.erase(unique(arr.begin(), arr.end()), arr.end());
  return arr;
}

template <typename T>
auto get_rank(const vector<T> &unique_sorted, const T &x) {
  return lower_bound(unique_sorted.begin(), unique_sorted.end(), x) -
         unique_sorted.begin();
}
```

能動的解 $O(n \log n)$

```cpp
vector<pair<int, int>> p;
void input() {
  int n;
  cin >> n;
  p.resize(n);
  vector<int> times;
  for (auto &[a, b]: p) {
    cin >> a >> b;
    times.emplace_back(a);
    times.emplace_back(b);
  }
  times = get_unique_sorted(move(times));
  for (auto &[a, b]: p) {
    a = get_rank(times, a);
    b = get_rank(times, b);
  }
}
```

```cpp
int main() {
  input();
  int N = p.size();
  vector<int> timeline(N, 0);
  for (auto [a, b] : p) {
    ++timeline[a];
    --timeline[b];
  }
  int ans = 0, pre = 0;
  for (int i = 0; i < N; ++i) {
    pre += timeline[i];
    ans = max(ans, timeline[i]);
  }
  cout << ans << endl;
  return 0;
}
```

更精簡的解

$O(n \log n)$

```cpp
int main() {
  int n;
  vector<pair<int, int>> v;
  cin >> n;
  for (int i = 0; i < n; ++i) {
    int l, r;
    cin >> l >> r;
    v.emplace_back(l, 1);
    v.emplace_back(r, -1);
  }
  int cur = 0, ans = 0;
  sort(v.begin(), v.end());
  for (auto [_, x] : v) {
    cur += x;
    ans = max(ans, cur);
  }
  cout << ans << "\n";
  return 0;
}
```

Two Pointer 爬行法

- 用 L, R 兩個 pointer 走過整個陣列
- 但過程中兩個 pointer 只能有遞增操作

CSES 1660 - Subarray Sums I

- 給你長度是 $n(n \leq 2 \cdot 10^5)$ 的正整數陣列 A
  以及整數 $x(0 < x \leq 10^9)$

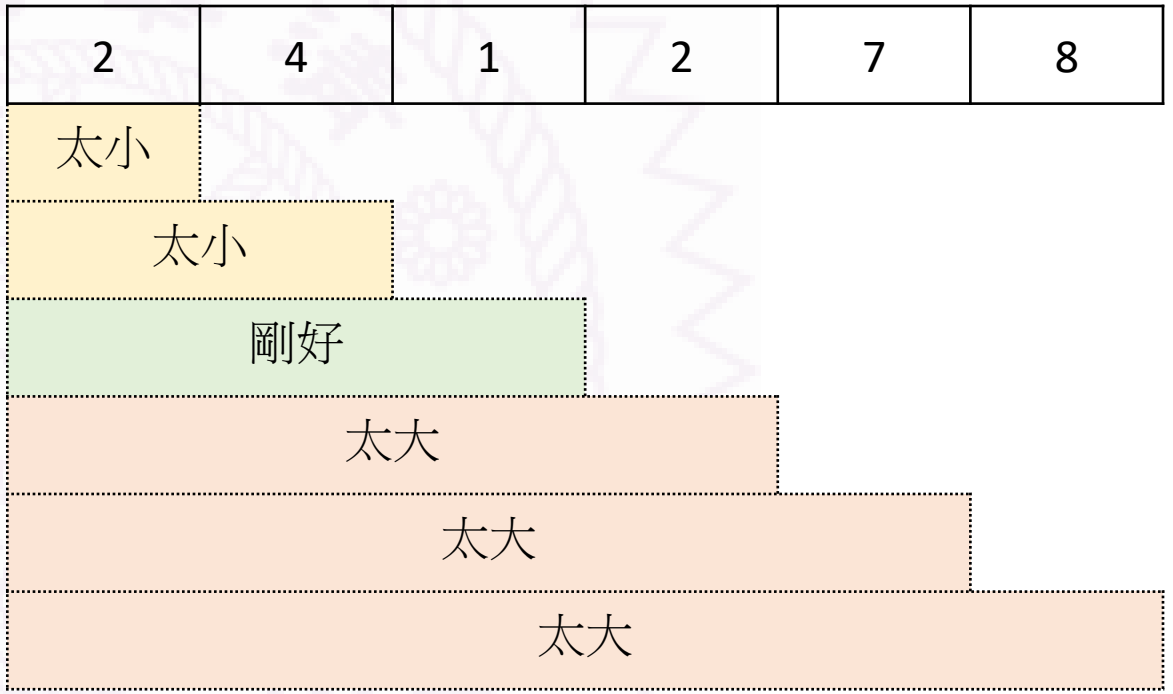
- 請你計算 A 中所有區間總合為 x 的區間有幾個

- 跟 Subarray Sums II 不一樣的地方是所有數字都是正的

CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

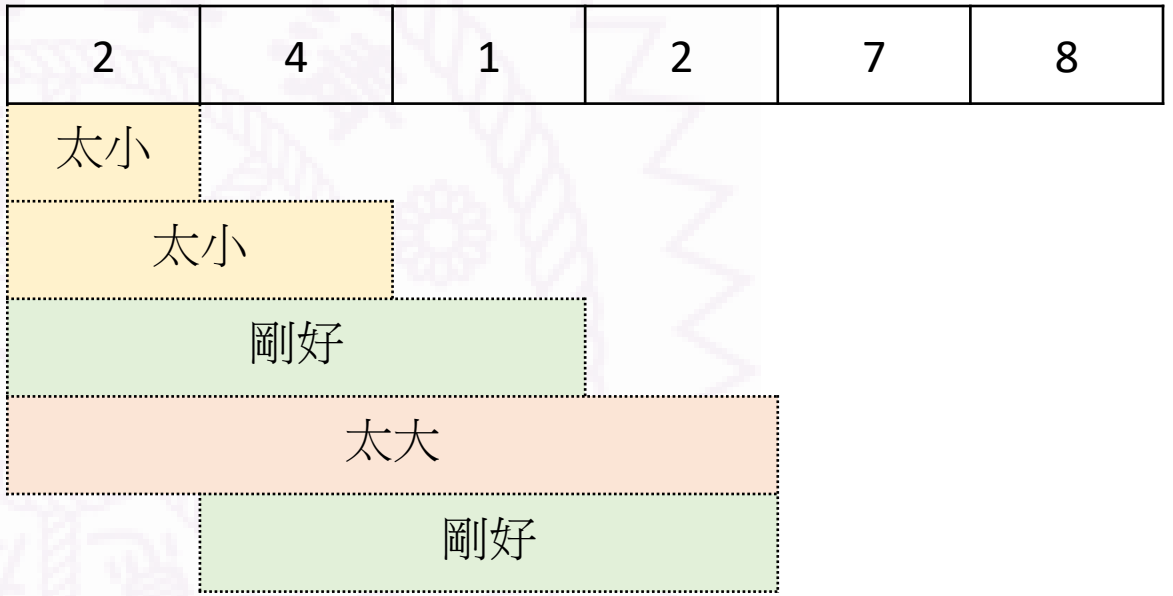


CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

- 一旦太大
  - 把左界往右一格

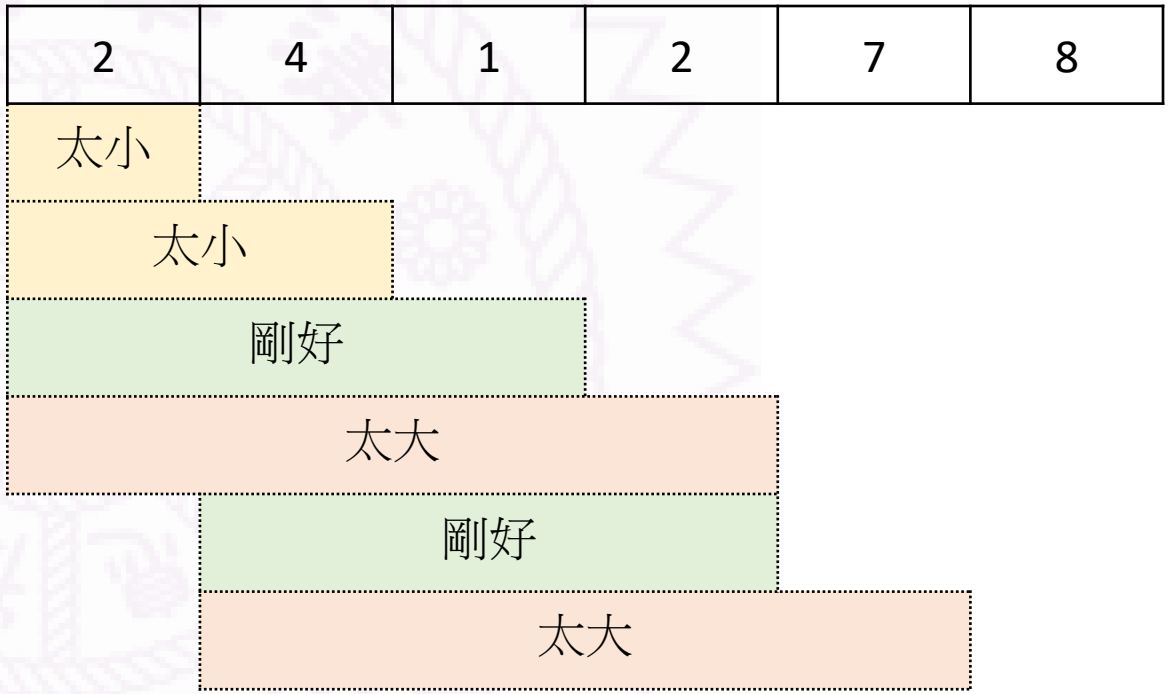


CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

- 一旦太大
  - 把左界往右一格

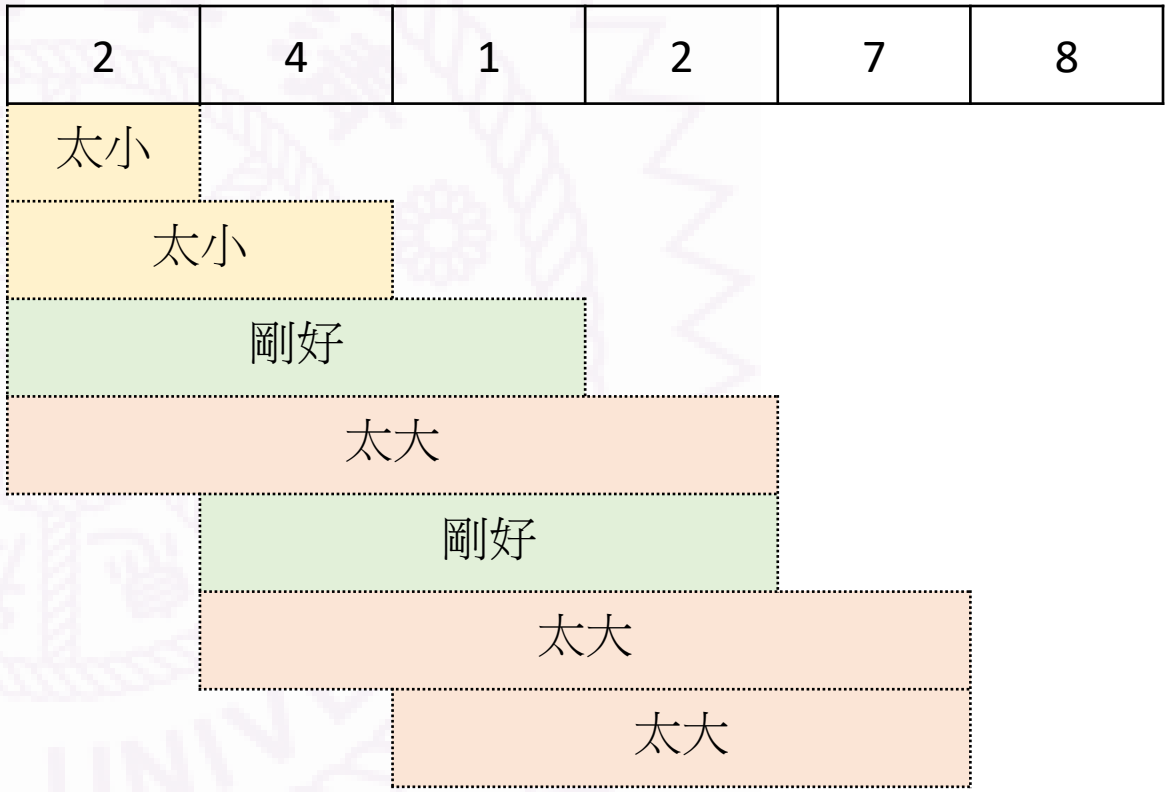


CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

- 一旦太大
  - 把左界往右一格

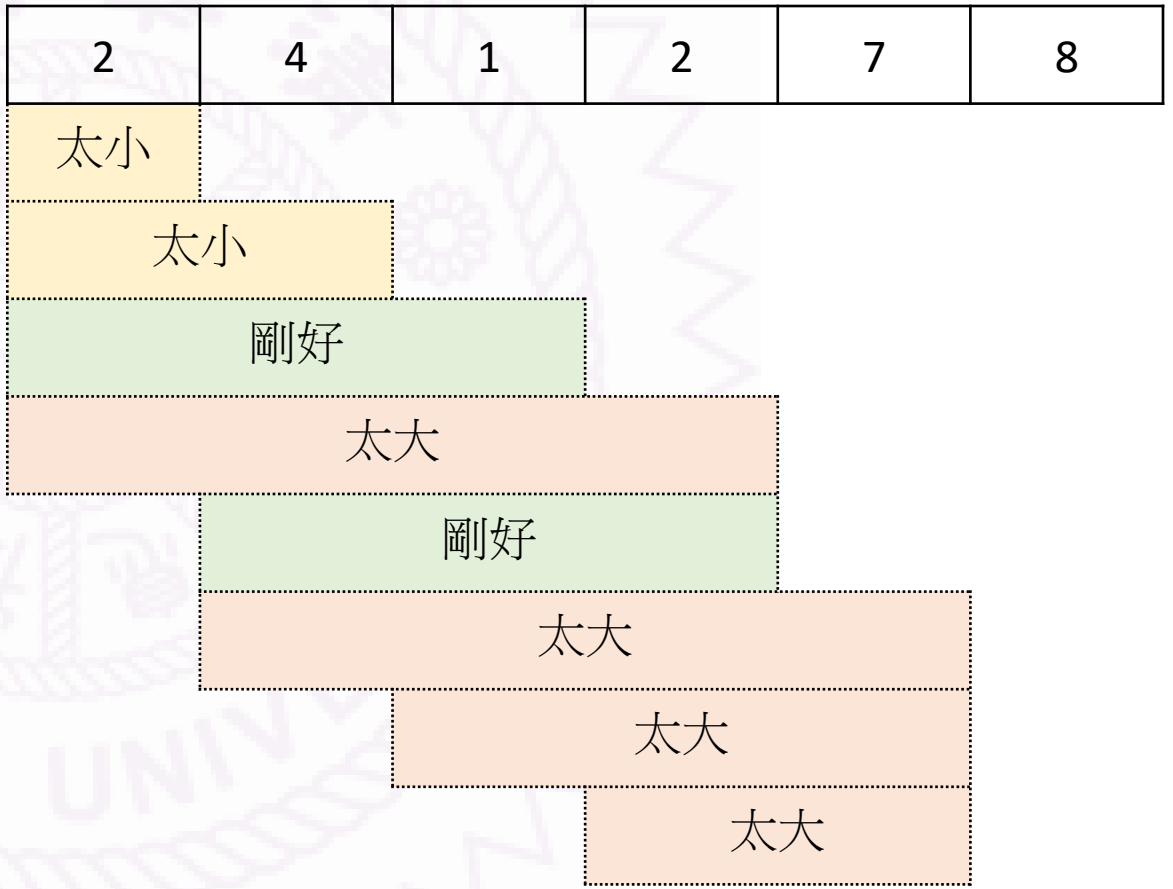


CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

- 一旦太大
  - 把左界往右一格

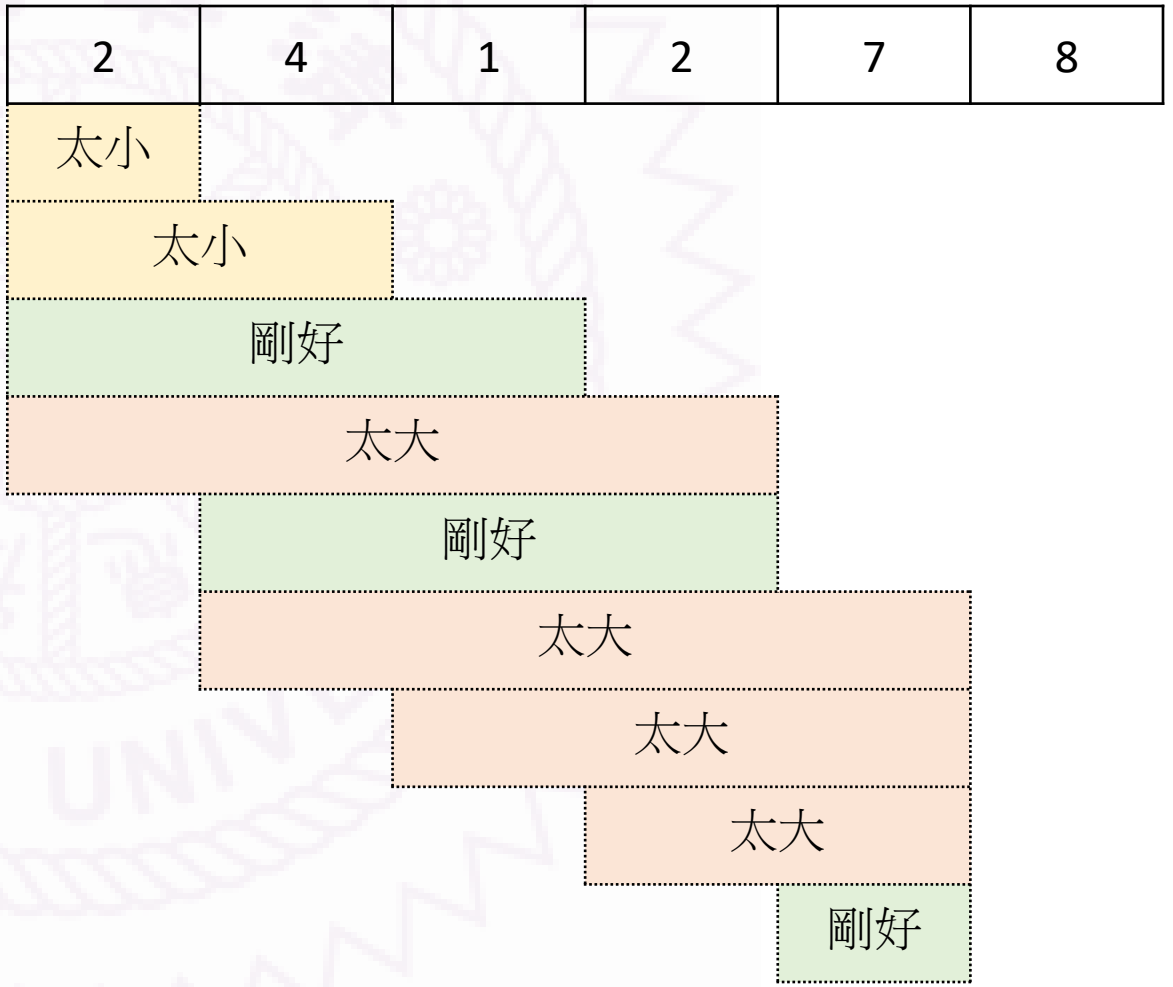


CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

- 一旦太大
  - 把左界往右一格

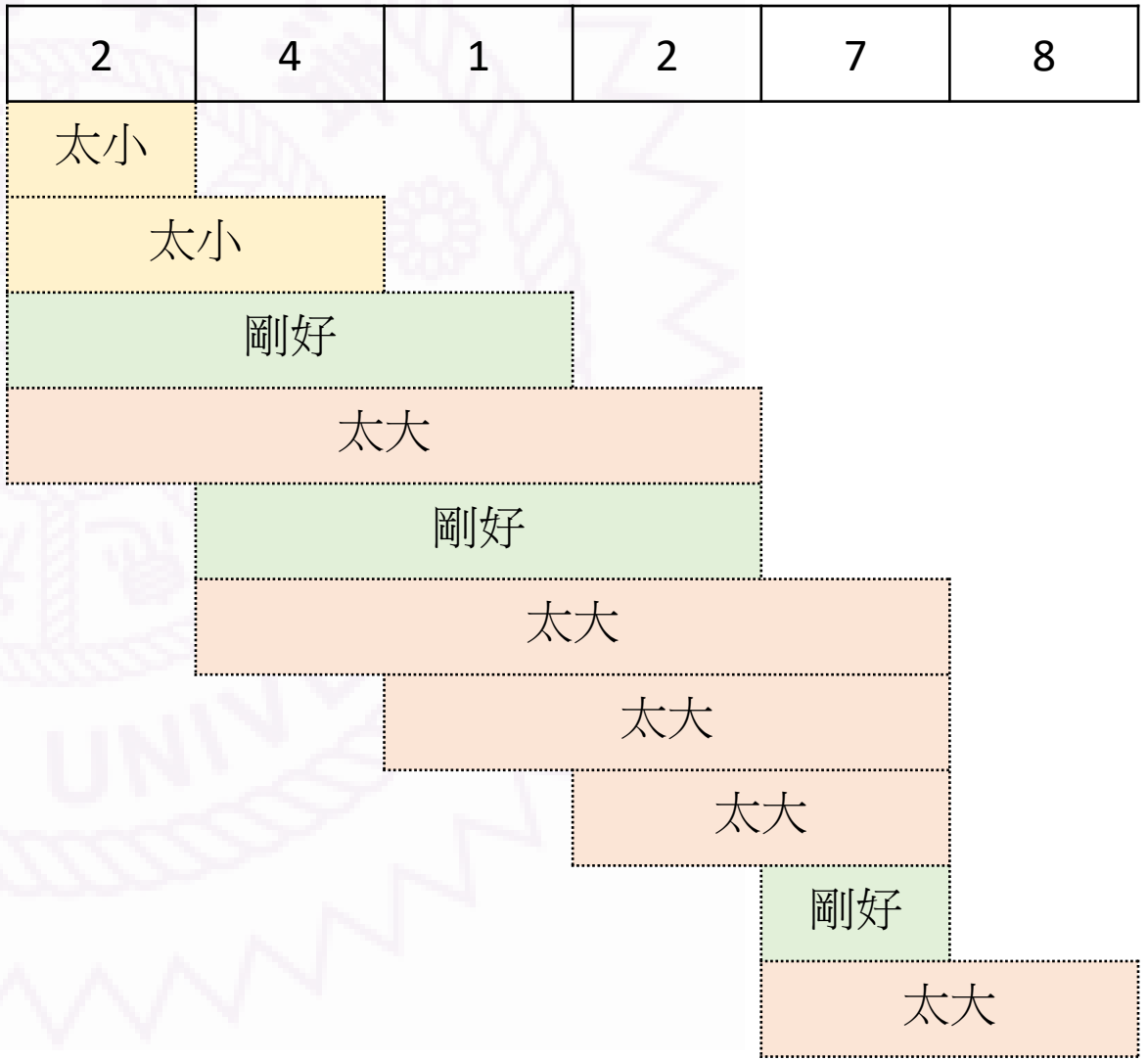


CSES 1660 - Subarray Sums I

- $x = 7, A = (2,4,1,2,7,8)$
- 觀察一下暴力枚舉的行為

- 右界持續增加，左界不變

- 一旦太大
  - 把左界往右一格



CSES 1660 - Subarray Sums I
關鍵程式碼 $O(n)$

```
int ans = 0;
long long cnt = 0;
for (int L = 0, R = 0; R < n; ++R) {
  cnt += A[R];
  while (cnt > x) {
    cnt -= A[L++];
  }
  if (cnt == x) {
    ++ans;
  }
}
```

CSES 1141 - Playlist 回家思考題

- 給你一個數字序列 $S = (s_1, s_2, \ldots, s_n)$

- 請你找出一個最長的子序列 $S[L, R]$
  使得 $s_L, s_{L+1}, \ldots, s_R$ 皆為不同數字

  - 如何套用 Two Pointer?
  - $[L, R]$ 夾的區間需要維護什麼性質？
  - 動 L, R 的規則？

CSES 1141 - Playlist 關鍵程式碼 $O(n \log n)$

```
int ans = -1;
map<int, int> cnt;
for (int L = 0, R = 0; R < n; ++R) {
  ++cnt[S[R]];
  while (cnt[S[R]] != 1) {
    --cnt[S[L++]];
  }
  ans = max(ans, R - L + 1);
}
```

CSES 1640/1641/1642 - Sum of 2/3/4 Values

- 試著用你剛剛學的技巧解決這些問題

Greedy 貪心法

- 有 n 桶冰淇淋，第 i 種口味有 w_i 公克，滿足度 v_i 每公克
- 已知甜筒最多能裝 m 公克，最多能裝的滿足度為多少？
  - 每種口味可以裝任意重量的冰淇淋，w_i, v_i, m 都是正整數

- ⇒ Ans: 從最好吃的開始裝，裝完換次好吃的 …

Greedy 貪心法

- 有 n 條薯條，第 i 條重量 w_i 公克，滿足度 v_i
- 已知紙盒最多能裝 m 公克，最多能裝多少滿意度的薯條？
  - 每次要裝完整的薯條, w_i, v_i, m 都是正整數

- ⇒ Ans: 每次挑**最大** $\left(\frac{v_i}{w_i}\right)$ **值**的開始裝 … ???

Greedy 貪心法使用規則

局部最佳導致全域最佳

- 每一步的最佳選擇，最終能導致整體最佳解。

最佳子結構（Optimal Substructure）

- 問題可以分解成子問題，且子問題的最佳解能組成原問題的最佳解。

貪心選擇性質（Greedy Choice Property）

- 一旦做出一個選擇，後續的選擇不會影響之前的選擇。

Greedy 貪心法

最佳冰淇淋選擇剛好符合三種性質

Greedy:
每次拿一克最好吃的冰淇淋
左邊的最佳解 = 右邊的最佳解 + 100

如何證明貪心法是否是最佳解呢?

- 反證法
  - Suppose not: 我不挑最好吃的冰淇淋 … Contradiction
  - ⇒ 我一定可以挑最好吃的冰淇淋

- 數學歸納法
  - Base case: 當我的甜筒大小只有 $m = 1$ 時，我挑最好吃的可以得到最佳解
  - Inductive case: 當我的甜筒大小有 $m = k (k > 1)$ 時
    假設 Greedy 可以得到 $m < k$ 時的最佳解
  - … ⇒ Greedy 也可以得到 $m = k$ 時的最佳解

演算法競賽中的 Greedy

考古題

- Greedy 的題目不好出
- 大量練習就會知道有哪些類型

直覺猜測

- 寫多了就會有
- 如果有時間再證明

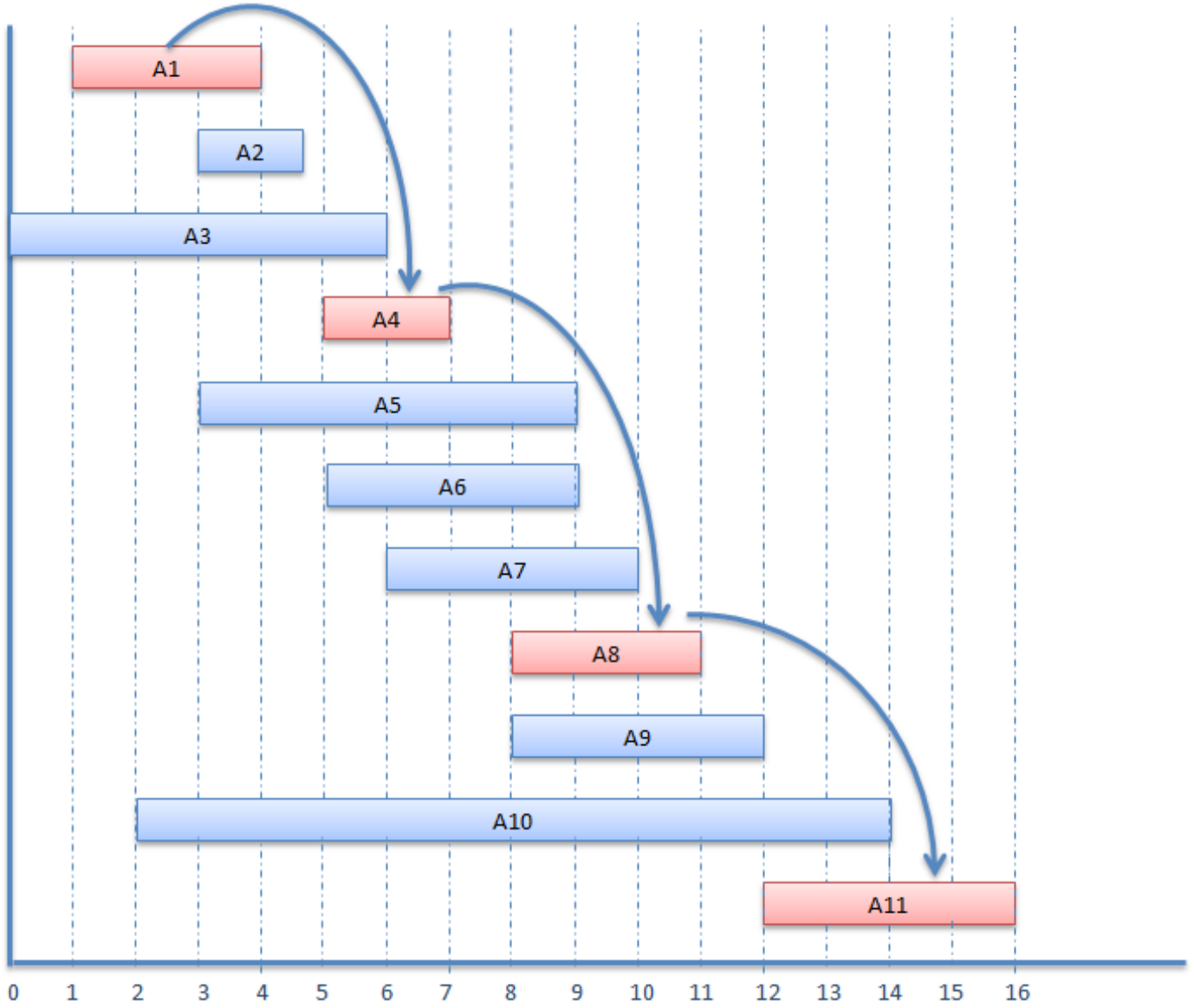
CSES 1629 - Movie Festival

- 有 n 場電影，第 i 場電影的開始時間是 s_i，結束時間是 e_i
- 你不可以同時看兩場電影
- 問你最多可以完整看完幾場電影

CSES 1629 - Movie Festival

- Greedy Algorithm:
  - 按照結束時間排序
  - 先挑最早結束的電影
  - 再挑剩下不重疊的電影中最早結束的

- 為什麼正確？
- ⇒ 去看《演算法導論》(CLRS) Ch 16.1



TIOJ 1072 . A.誰先晚餐

- 經典練習題
- https://tioj.ck.tp.edu.tw/problems/1072

UVa 10954 - Add all

- 給定 n 個數字
- 每次將兩個數字 a, b 合併為 $(a + b)$，並須付出成本 $(a + b)$
- 求合併 n 個數字成為一個數字所需最低成本

UVa 10954 - Add all

- Greedy Algorithm:
  - 每次挑最小的兩個數字合併
  - 用 min heap 可以很輕鬆實作出來

- 證明:
  - 請參考 Huffman 編碼

Greedy?

- 有 n 顆蘋果，第 i 顆重量 w_i 公克，滿足度 v_i
- 你想從中挑恰好 k 棵蘋果 (你一定要挑整顆，不能切，不能榨汁)
- 使得最後 CP 值 (= 滿足度加總 / 重量加總) 最大

分數規劃 (0 - 1 Fractional Programming)

- 给出 a_i 和 b_i，求一组 $w_i \in \{0,1\}$，最小化或最大化：

$$\frac{\sum_{i=1}^{n} a_i \cdot w_i}{\sum_{i=1}^{n} b_i \cdot w_i}$$

- 有些題目會有奇怪的性質，例如
  - $w_i = 1$ 的要剛好 k 個

- 直接計算這種有分數的最佳值不太好算，我們想辦法**去除分數**

分數規劃

Binary Search 找使得

$$\sum w_i(a_i - g \cdot b_i) \geq 0$$

最大的 g

- 假設是最大化問題，已知數字 g 滿足：

$$\frac{\sum a_i \cdot w_i}{\sum b_i \cdot w_i} \geq g$$

$$\Rightarrow \left(\sum a_i \cdot w_i\right) - g \cdot \left(\sum b_i \cdot w_i\right) \geq 0$$

$$\Rightarrow \sum w_i(a_i - g \cdot b_i) \geq 0$$

- 如果等號左邊的最大值比 0 大，表示 g 比最佳解小

完整程式碼

```cpp
double solve(int n, int k, const vector<double> &w, const vector<double> &v) {
  double left = 0, right = 1e9, ans = 0;
  vector<int> index(n);
  iota(index.begin(), index.end(), 0);
  while (right - left > 1e-6) { // 精確度
    double mid = (left + right) / 2, sum_h = 0, sum_v = 0, sum_w = 0;
    auto h = [&](int i) { return v[i] - mid * w[i]; };
    sort(index.begin(), index.end(), [&](int i, int j) { return h(i) > h(j); });
    for (int i = 0; i < k; ++i) {
      sum_h += h(index[i]);
      sum_v += v[index[i]];
      sum_w += w[index[i]];
    }
    if (sum_h >= 0) {
      ans = sum_v / sum_w;
      left = mid;
    } else
      right = mid;
  }
  return ans;
}
```