貪心演算法
Greedy Algorithm

2024 南九校聯合寒訓 - 演算法組

講師：陳俊安 Colten

2024.02.06

陳俊安 Colten

- 國立成功大學資訊工程學系大二 (特選甲組)
- 專長：演算法 (Algorithm)
- 現役競技程式設計選手
- ICPC 國際大學生程式設計競賽
  - 台灣桃園站 - 銅牌獎
  - 日本橫濱站 - 榮譽獎
- 2023 Hp Code Wars 全國程式設計爭霸戰 (大專組) 第六名
- 嘉義女中數理資優班獨立研究課程 - 資訊組指導老師

貪心演算法 Greedy Algorithm

- 貪心是一種大家都會的本能，而且出現在你我的日常生活中
- 下面這個情境題大家思考看看：
  - 現在你是超商店員，你要找零錢給顧客，為了避免顧客不耐煩，你使用來找零的貨幣數量必須越少越好，你應該怎麼做？

貪心演算法 Greedy Algorithm

- 貪心是一種大家都會的本能，而且出現在你我的日常生活中
- 下面這個情境題大家思考看看：
  - 現在你是超商店員，你要找零錢給顧客，為了避免顧客不耐煩，你使用來找零的貨幣數量必須越少越好，你應該怎麼做？
    - 大部分的大家都會有的直覺：
    - 能用大面額的貨幣就盡可能的用大面額的

貪心演算法 Greedy Algorithm

- 如果我們要找零 1138 元
  - 1000 * 1
  - 100 * 1
  - 10 * 3
  - 1 * 8

貪心演算法 Greedy Algorithm

- 相信這個策略對大家來說應該都還算很直覺的策略
- 但是如果有人問你為什麼這個策略一定會對，你能說服ㄇ？
- 我們挑一個同學來講看看：D

貪心演算法 Greedy Algorithm

- 相信這個策略對大家來說應該都還算很直覺的策略
- 但是如果有人問你為什麼這個策略一定會對，你能說服ㄇ？
  - 以台灣的貨幣來說，我們任意取兩個貨幣面額 a, b (a < b)
  - 都滿足 a | b (b 能被 a 整除) 的性質
  - 也就是說我如果要用 b / a 個面額為 a 的貨幣，不如直接使用 1 個面額為 b 的貨幣

貪心演算法 Greedy Algorithm

- 剛剛那樣子的策略，雖然很直覺，但這就是一種貪心演算法
- 所以現在大家都會貪心算法了吧，~~那好像可以下課了！~~
- 不過有時候情境只要稍微換一下，我們的直覺就會是錯的哦...
- 我們來看一個例子！

貪心演算法 Greedy Algorithm

- 假設有一個國家的貨幣種類只有三種 1, 23, 50
- 今天必須找零 58 元，用我們剛剛的策略，會發生什麼事情？

貪心演算法 Greedy Algorithm

- 假設有一個國家的貨幣種類只有三種 1, 23, 50
- 今天必須找零 58 元，用我們剛剛的策略，會發生什麼事情？
  - 我們必須使用：50 * 1 + 1 * 8
  - 但最佳策略為：23 * 2 + 1 * 2
  - 剛剛的策略錯了耶...，為什麼？

貪心演算法 Greedy Algorithm

- 還記得我們原本的題目有提到一個說服別人的方法ㄇ？
  - 我們任意取兩個貨幣面額 a, b (a < b)
  - 都滿足 a | b (b 能被 a 整除) 的性質
- 我們的貨幣變成了 1, 23, 50，就不滿足這個性質了
  - 50 不能被 23 整除
- 因此這樣子的策略在這個情況下是不成立的

貪心演算法 Greedy Algorithm

- 剛剛提到的題目都是非常經典的問題 - 找零問題
- 第一個版本的題目我們已經知道可以用貪心演算法解決
- 第二個版本的題目其實也是有方法可以做的 (剛好是下一節課的東西)
  - 動態規劃 Dynamic Programming
  - 這兩個演算法常常被使用來解決最佳化問題，也有許多相似的地方

貪心演算法 Greedy Algorithm

- 從剛剛的例子可以發現，直覺有時候不一定是對的
- 所以如何驗證我們的直覺或想法就變得非常重要
- 剛剛為什麼會請大家提出一個說服別人的方法？
  - 你必須有一個合理的理由，才能驗證你的想法是不是對的
  - 而驗證個這個過程，我們也叫做證明

貪心演算法 Greedy Algorithm

- 這堂課會有一點點數學的成分在，但不用太擔心
- 我們先來看看貪心題目的解題步驟好了！

貪心演算法 Greedy Algorithm

- 觀察：
  - 觀察性質，貪心最關鍵的一步，抓住關鍵性質來輔助我們提出一個“可能”的策略
- 嘗試：
  - 提出一個策略後，試著思考在一些不同的情境中，這個策略有沒有可能會失敗
- 驗證：
  - 以實際的數學證明驗證此策略

例題 1 - CSES Problem Set Movie Festival

- 有 n 部電影，給定每一部電影的開始時間與結束時間，在看其中一部電影的時候不能同時看其他部，請問在最佳策略下，一個人最多可以看幾部電影？

例題 1 - CSES Problem Set Movie Festival

- 有 n 部電影，給定每一部電影的開始時間與結束時間，在看其中一部電影的時候不能同時看其他部，請問在最佳策略下，一個人最多可以看幾部電影？
- 觀察：
  - 如果要看越多的電影，感覺... 看越早的電影越好！

例題 1 - CSES Problem Set Movie Festival

- 有 n 部電影，給定每一部電影的開始時間與結束時間，在看其中一部電影的時候不能同時看其他部，請問在最佳策略下，一個人最多可以看幾部電影？
- 觀察：
  - 如果要看越多的電影，感覺... 看越早的電影越好！
- 嘗試：
  - 你能提出一個這個策略會失敗的情況ㄇ？

例題 1 - CSES Problem Set Movie Festival

- 有 n 部電影，給定每一部電影的開始時間與結束時間，在看其中一部電影的時候不能同時看其他部，請問在最佳策略下，一個人最多可以看幾部電影？
- 因為剛剛的策略失敗了，我們重新思考一個
- 觀察：
  - 既然要看越多部電影，我們每一部電影的結束時間變得很重要，因為只要結束時間越早我們繼續看越多電影的機會越多

例題 1 - CSES Problem Set Movie Festival

- 觀察：
  - 既然要看越多部電影，我們每一部電影的結束時間變得很重要，因為只要結束時間越早我們繼續看越多電影的機會越多
- 嘗試：
  - 試了一下你會發現，這個策略好像還不錯耶 ><

例題 1 - CSES Problem Set Movie Festival

- 礙於時間有限無法好好的講證明 + 我知道大家討厭數學
- 所以我們理性的分析一下：
  - 只要你盡量選那些結束時間越早的電影表示：
    - 你接下來能繼續看下一部電影的機會越多
    - 就有點像是你離集合時間越早，能做的事情越多的概念

例題 2 - TOI 2009 資訊奧林匹亞初選第三題

- 現在有 n 本書，每一本書必須先印刷才能裝訂
- 你有 1 台印刷機，n 台裝訂機
- 印刷機一次只能印一本書
- 裝訂機一次只能裝訂一本書
- 給定每一本書所需要的印刷時間跟裝訂時間
- 請你找到所有書都完成裝訂所需要的最少時間
- 1 <= n <= 1000 (但其實可以再更大)

例題 2 - TOI 2009 資訊奧林匹亞初選第三題

- 我們先觀察一下，我們有 n 台裝訂機，而我們的書是 n 本
- 這表示不管什麼時候都有裝訂機可以用

例題 2 - TOI 2009 資訊奧林匹亞初選第三題

- 每一本書：印刷 -> 裝訂
- 印刷機只有一台，所以不管怎麼樣，我們的答案一定不會小於所有書的印刷時間加總
  - 印刷機只能一本一本慢慢印刷
  - 如果有 5 本書的印刷時間分別是 1,2,3,4,5
  - 所有書都裝訂完的時間一定 >= 1 + 2 + 3 + 4 + 5

例題 2 - TOI 2009 資訊奧林匹亞初選第三題

- 其實從這邊也可以發現，每一本書的先後順序不影響總印刷時間
- 關鍵就在於我們什麼時候把書拿去裝訂
- 觀察到這邊，有想到策略了ㄇ？
- 我們抽一個同學來講講看好了

例題 2 - TOI 2009 資訊奧林匹亞初選第三題

- 有一個很直覺的想法，需要被裝訂越久的書，我們越應該先做
- 所以我們只要照著裝訂時間排序好，最佳的順序就出來了
- 我們就可以在 O(NlogN) 的時間做完這一題

例題 2 - TOI 2009 資訊奧林匹亞初選第三題

- 如果你喜歡數學，我們把式子列出來！
- 我直接用寫的：

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- 有 n 台機器排成一直線，且有 m 個工作
- 每一台機器生產出 1 個資料需要花費 t[i] 的時間
- 由左邊開始編號每一台機器，第 i 個工作必須使用區間 [l , r] 的機器，且每一台機器必須分別產出 w[i] 的資料
- 你可以重新排列所有機器，請找出最小需要花費的時間

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- 因為我們可以任意的排列，所以我們先好好計算出每一個編號的機器需要產生的資料數量
- 如果有一個工作指定使用編號 [2 , 4] 的機器，且需要 10 單位的資料
- 我們維護一個序列，這個序列紀錄各個編號的機器需要生產的單位
- 如果有 5 台機器：{ 0 , 10 , 10 , 10 , 0 }

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- 又有一個工作指定 [1 , 3]，要產生 7 單位的資料
- 序列會變成 { 7 , 17 , 17 , 10 , 0 }
- 假設沒有其他工作了，那我們就得出了所有編號的機器需要負責的資料數量，那接下來的問題就變成如何分配機器了
- 很直覺的，分配機器的時候，需要負責越多資料的位置，我們要使用速度越快的機器

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- 如此一來這一題的做法就出來了，但還有一個問題
- 一開始我們維護每一個編號負責的資料數量的那一個序列應該怎麼求
- 第一個很顯然的做法，一開始開好一個陣列，並且用迴圈把區間的每一個位置跑過一次，一個一個加
- 但這樣太慢了，時間複雜度會變成 O(N^2)
- 有人想的到更好的做法ㄇ？

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- 假設目前有一個任務要用到區間 [2 , 4] 且需要生產 10 單位的資料
- 表示從 2 開始，會多 10 個任務
- 從 5 開始，會少 10 個任務

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- 我們額外開一個陣列 b 紀錄每一個機器所需的生產變化量
- b[i] = 從編號 i 的機器開始，接下來的機器會需要多生產 b[i] 單位的資料
- Example：
  - [1 , 5] 生產 10
  - [2 , 3] 生產 7
  - b[1] = 10, b[2] = 7, b[4] = -7, b[6] = -10

例題 3 - APCS 2021 年 10 月第三題 - 生產線

- Example：
  - [1 , 5] 生產 10
  - [2 , 3] 生產 7
  - b[1] = 10, b[2] = 7, b[4] = -7, b[6] = -10
- 接下來我們從左到右將所有的變化量加總起來，就會是編號 i 這台機器所需要生產的資料數量
- 如此一來，計算區間的時間複雜度就從 O(N^2) -> O(N)

例題 3 - APCS 2021 年 10 月第三題 - 生產線

```
16      for(int i=0;i<m;i++)
17      {
18          int l,r,k;
19          cin >> l >> r >> k;
20          check[l] += k;
21          check[r+1] -= k;
22      }
```

例題 3 - APCS 2021 年 10 月第三題 - 生產線

```
24    vector <int> a,b;
25    for(int i=0;i<n;i++)
26    {
27        int input;
28        cin >> input;
29        a.emplace_back(input); // 每一台機器的生產量
30    }
31
32    int total = 0;
33    for(int i=1;i≤n;i++)
34    {
35        total += check[i];
36        if( total > 0 ) b.emplace_back(total);
37    }
38
39    sort(a.begin(),a.end());
40    sort(b.rbegin(),b.rend());
41
42    int ans = 0;
43    for(int i=0;i<b.size();i++)
44    {
45        ans += b[i] * a[i];
46    }
47
48    cout << ans << "\n";
```

列式觀察貪心：CSES Problem Set Tasks and Deadlines

- 你有 n 個任務，完成第 i 個任務的所需時間為 a_i，期限為 d_i
- 每一個任務都必須完成，如果完成第 i 個任務的時間是 f
- 那麼你可以得到 f - d_i 的獎勵
- 請你設計一個程式計算出在最佳排程策略下，可以拿到多少獎勵

列式觀察貪心：CSES Problem Set Tasks and Deadlines

- 我們先不管策略，假設順序是給定的，如何計算答案？
- 任務 1：a = 10, d = 15
- 任務 2：a = 7, d = 22
- 假設先做任務 1 再做 2
  - Ans = (15 - 10) + (22 - (10 + 7))

列式觀察貪心：CSES Problem Set Tasks and Deadlines

- 接下來我們嘗試列出式子觀察
- 這邊假設我們的任務順序是 p_1 ~ p_n

列式觀察貪心：CSES Problem Set Tasks and Deadlines

- 因此我們只要利用我們剛剛發現的策略就可以算出最佳答案了
- 時間複雜度：O(NlogN)

```
26    int n;
27    vector <pair<int,int>> a;
28    cin >> n;
29
30    for(int i=0;i<n;i++)
31    {
32        int x,y;
33        cin >> x >> y;
34        a.emplace_back(make_pair(x,y));
35    }
36
37    sort(a.begin(),a.end(),cmp);
38
39    int ans = 0,now = 0;
40
41    for(int i=0;i<n;i++)
42    {
43        now += a[i].f;
44        ans += ( a[i].s - now );
45    }
46
47    cout << ans << "\n";
```

列式觀察貪心：CF 553 pD. Stas and the Queue at the Buffet

- 有 n 個人，每一個人有兩個數值 a_i, b_i
- 現在要幫這 n 個人排隊
- 如果第 i 個人被排在 j 這個位置，他的不滿意度會是：
  - a_i * (j - 1) + b_i * (n - j)
- 請你設計一個程式找出最佳的排隊方法下，所有人的不滿意度總和最小會是多少

列式觀察貪心：CF 553 pD. Stas and the Queue at the Buffet

- 我們一樣列出式子來看看吧！
- 然後你就會發現這題 1600 的題目是水題

經典問題 - 最大連續和

- 給定一個長度為 n 的序列，請找出一個區間，這個區間的和要越大越好
- 相信大家一定都能想到 O(n^2) 的作法，但這太遜了
- 我們來觀察一個 Case：
- 10, 20, -40, 10, 40
- 我們可以發現一個很重要的原則，只要數字 >= 0，沒有不拿他的道理

經典問題 - 最大連續和

- 10, 20, -40, 10, 40
- 我們從左邊開始盡量拿數字直到 -40 時遇到問題
- 如果我們拿了 -40，總和會變成 10 + 20 + (-40) = -10
- 當我們拿的數字和 < 0 時表示，什麼東西都不拿還更好
- 這點是非常重要的觀察，所以我們可以總結我們的策略：
  - 盡量拿數字，當拿到總和 < 0，清空我們之前所有拿的東西
  - 一邊做一邊更新答案

經典問題 - 最大連續和

```
25      int n;
26      cin >> n;
27      vector<int>a(n);
28
29      long long ans = -1e18;
30      for(int i=0;i<n;i++) cin >> a[i], ans = max(ans,a[i]);
31
32      long long total = 0;
33
34      for(int i=0;i<n;i++)
35      {
36          total += a[i];
37          if( total < 0 ) total = 0;
38          ans = max(ans,total);
39      }
40
41      cout << ans << "\n";
```

變化應用：CF EDU123 pC. Increase Subarray Sums

- 給定一個長度為 n 的序列，與一個非負整數 x
- 每一次操作可以把某個位置的值增加 x
- 定義 f(k) 表示你可以把 k 個不同的位置增加 x 後的最大連續和
- 請你設計一個程式求出 f(0) ~ f(n)
- 1 <= n <= 5000

變化應用：CF EDU123 pC. Increase Subarray Sums

- 這題比較難一點點，我們先做一些觀察
- 假設某個區間是最大連續和的區間，這個區間的長度是 10，且最大連續和是 48763
- 那麼如果你可以選 3 個不同的位置增加 x，最大連續和的結果會變成 48763 + 3x

變化應用：CF EDU123 pC. Increase Subarray Sums

- 有了這一點觀察之後我們可以得出一個小小的方向
- 我們根本不用在乎我們到底選了什麼點，把他們加上 x
- 所以剩下的問題就只有我們的最大連續和答案會落在哪一個區間了

變化應用：CF EDU123 pC. Increase Subarray Sums

- 對於 f(0) ~ f(n) 來說，這一個最大連續和一定落在某一個區間上
  - 這是廢話
- 用我們的觀察思考看看做法：
  - 對於 f(0) ~ f(n)，我們要算的 f(k) 的時候都去枚舉所有區間的 L 跟 R
  - 計算出他的最大連續和之後，將結果加上 kx
  - 但好像怪怪的？

變化應用：CF EDU123 pC. Increase Subarray Sums

- 對於 { 10, 20, -100, -200 }, x = 7 來說：
- 當計算 f(4) 時，我們發現最大連續和為 10 + 20 = 30
- 30 + (4 * 7) = 58 是錯誤的答案
  - { 17, 27, -93, -193 } 的最大連續和為 17 + 27 = 44
- 有發現問題出現在什麼地方ㄇ？
- 我們雖然求出最大連續和了，但我沒辦法知道我這一個最大連續和的區間長度，有可能我使用了很少的元素，就會使答案發生問題

變化應用：CF EDU123 pC. Increase Subarray Sums

- 除此之外，我們也可以發現這樣的做法時間複雜度會是 O(n^3)
- 因此我們必須想想其他辦法

變化應用：CF EDU123 pC. Increase Subarray Sums

- 我們剛剛遇到的問題是：無法確定最大連續和的區間長度
- 只要這個問題解決了，這題就搞定了
- 如果你想到了一個可以用資料結構做到的方法那先請你不要衝動
- 我們有更讚的作法

變化應用：CF EDU123 pC. Increase Subarray Sums

- 我們一開始先建立一個陣列 b
- b[i] 表示區間長度為 i 的最大連續和是多少
  - 我們一開始先預處理好這一個表格
- 接下來計算 f(0) ~ f(n)
  - 最佳的答案一定會是某一個區間 = 區間長度 0 ~ n 其中一個
  - 枚舉所有區間長度，把該長度的最大連續和拿來更新答案
  - 對於區間長度 i 來說：
    修改 k 次的最大連續和會是 b[i] + min(i,k) * x

變化應用：CF EDU123 pC. Increase Subarray Sums

```
vector<int>a(n+1);
for(int i=1;i<=n;i++) cin >> a[i];
for(int i=1;i<=n;i++)
{
        int total = 0;
        for(int k=i;k<=n;k++)
        {
                total += a[k];
                dp[k-i+1] = max(dp[k-i+1],total);
        }
}

for(int i=0;i<=n;i++)
{
        int ans = 0;
        for(int k=1;k<=n;k++) ans = max(ans,dp[k]+min(k,i)*m);

        cout << ans << " ";
}
cout << "\n";
```