練習題1~5題解、6~11提示

附虛擬碼

1. 握手宴會 Subtask 1~3

- 如果兩個人有握過手，就把它們的pair丟到set裡面（C++的<set>）
- 查詢也直接在set查

- 注意因為握手是無序的但pair是有序的，所以丟進去前要先限定 $a \leq b$，查詢的時候也一樣
- MLE 50

1. 握手宴會

- 跟HW2一樣，排序+二分搜就可以了！
- AC 100

```
pair<int, int> arr[M]; // 所有握手記錄

for (pair(a, b) in arr) {
  if (a > b) swap(a, b);
}
sort(arr);

for (pair(a, b) in query) {
  if (a > b) swap(a, b);
  if (binary_search(arr, (a, b))) puts("yes");
  else puts("no");
}
```

2. P-蹺蹺板 Subtask 1-2

- 直覺的作法：把x從0枚舉到大
- 枚舉所有支點的位置，算力矩檢查能不能平衡
- $O(N^3)$，TLE 20

2. P-蹺蹺板 Subtask 3

- 在同一個組合之下，枚舉支點的時候算力矩一定要每次重算嗎?
- 觀察如果把支點從P移到$P+1$，兩邊的力矩會如何變化?

- 左邊的力矩增加$a_0 + \cdots + a_P$，右邊的力矩減少$a_{P+1} + \cdots + a_{N-1}$
- 礙於篇幅限制，接下來的投影片一律使用$\sum_{i=x}^{y} a_i$表達$a_x + \cdots + a_y$，請自行理解(?

2. P-蹺蹺板 Subtask 3

- 注意到$\sum_{i=P+1}^{N-1} a_i = \sum_{i=0}^{N-1} a_i - \sum_{0}^{P} a_i$

- 先算好$\sum_{i=0}^{N-1} a_i$
- 對每一個x，一開始花$O(N)$算好一開始的力矩，因為$\sum_{0}^{P} a_i$可以一邊枚舉一邊算，就可以花$O(N)$枚舉所有支點
- 總複雜度降為$O(N^2)$，TLE 30

2. P-蹺蹺板 Subtask 4

- 如果支點擺得太左邊，左邊力矩會小於右邊力矩；擺太右邊，右邊力矩小於左邊力矩
- 可以二分搜，$O(\log N)$找出對於一個x，支點該擺在哪裡（或者不存在支點）
- 但是要怎麼$O(1)$算出某一邊的力矩呢?
  - 列式!
- 給定交換狀況x與支點位置P，左邊力矩等於 $\sum_{i=0}^{x-1}(P-i)a_{N-i-1}+\sum_{i=x}^{P-1}(P-i)a_i$
  - （這裡假設$x<P$）

2. P-蹺蹺板 Subtask 4

- $\sum_{i=0}^{x-1}(P-i)a_{N-i-1} = \sum_{i=N-x}^{N-1} ia_i - (N-P-1)\sum_{i=N-x}^{N-1} a_i$
- $\sum_{i=x}^{P-1}(P-i)a_i = P\sum_{i=x}^{P-1} a_i - \sum_{i=x}^{P-1} ia_i$
- 所以現在的目標變成$O(1)$算出$\sum_{i=x}^{y} a_i$和$\sum_{i=x}^{y} ia_i$
  - 右邊的力矩也可以用這兩個表示
- 注意到$\sum_{i=x}^{y} a_i = \sum_{i=0}^{y} a_i - \sum_{i=0}^{x-1} a_i$，所以可以在最一開始的時候用$O(N)$算出所有的$\sum_{i=0}^{y} a_i$和$\sum_{i=0}^{y} ia_i$存起來，這樣就可以$O(1)$算出力矩
- 套用二分搜，總複雜度達到$O(N\log N)$，TLE 65

2. P-蹺蹺板

- 回到Subtask 3
  - 「左邊的力矩增加$a_0 + \cdots + a_P$，右邊的力矩減少$a_{P+1} + \cdots + a_{N-1}$」
- 那為甚麼不改算「右邊力矩-左邊力矩」呢?
- 每往右邊移一格，右邊力矩-左邊力矩就減少$a_0 + \cdots + a_{N-1}$
- 也就是說，算出支點擺最左邊的總力矩後，一個除法（$O(1)$）就知道支點要擺在哪裡了！
  - 如果能整除代表可以平衡

2. P-蹺蹺板

- 剩下的問題是對於每個x，要怎麼$O(1)$算出支點擺最左邊的力矩是多少
- 對於$x = 0$先用$O(N)$算出來
- x增加1的時候只是把兩個人交換而已，簡單計算一下，總力矩會增加$(N - 2x - 1)(a_x - a_{N-x-1})$

- $O(N)$，AC

2. P-蹺蹺板

```
torque = 0;
sum = 0;
for (i = 0; i < N; i++) {
  sum += a[i];
  torque += i * a[i];
}
for (x = 0;; x++) {
  if (torque % sum == 0) {
    print(x, torque / sum); exit;
  }
  torque += (N - 2*x - 1) * (a[x] - a[N - x - 1]);
}
```

3. 排隊買飲料 Subtask 1~4/5

- 只要店員一有空就立刻服務下一個客人，時間就一定會最短

- 開一個店員的陣列，代表這個店員要到甚麼時候才有空
- 每次挑最小的那個店員服務下個客人，並更新當前有空時間的值
- $O(N^2)$，TLE 40/52

3. 排隊買飲料

- 改成用priority_queue儲存每個店員的有空時間
- $O(N \log N)$，AC

```
priority_queue<int, vector<int>, greater<int>> pq;
push M zeros into pq;
for (i = 0; i < N; i++) {
  tmp = pq.top();
  pq.push(tmp + a[i]);
}
pop M-1 times;
print(pq.top());
```

4. Bonus Time Subtask 1

- 這個subtask字串長度只有13
- 枚舉其中一個字串所有的子序列，然後看看它能不能在另外一個字串上被湊出來（且段數不超過k）
- 對所有可行的解法求最大值
- $O(N2^N)$，TLE 15

4. Bonus Time Subtask 2~3

- DP!
- 這題的DP狀態比較難想
- 我們令$dp(i,j,k)$代表「第一個字串只留前i個字元、第二個字串只留前j個字元且最多只能取k個子字串的答案」
- 例如對於範例測資，$dp(3,3,1)=4$，因為兩人的字串分別是aab和aaa，兩個人可以都選aa，答案為4

- 可以先停在這裡想想看，你能寫出$dp(i,j,k)$的遞迴式嗎?

4. Bonus Time Subtask 2~3

- 可是這樣沒辦法判斷如果字串增加了一個字元，要不要另外多切一段
- 如果兩個人的最後一段都有選到最後一個字元，那就可以不用多切一段

- 所以我們令$dp2(i,j,k)$為「第一個字串只留前i個字元、第二個字串只留前j個字元，最多只能取k個子字串且兩個人的最後一段都有選到最後一個字元的答案」
- 這樣應該就可以寫出遞迴式了喔，可以再停下來想想看

4. Bonus Time Subtask 2~3

- 如果$a_{i-1} \neq b_{j-1}$，$dp(i,j,k) = \max\{dp(i,j-1,k), dp(i-1,j,k)\}$；$dp2(i,j,k) = 0$
  （因為既然兩個字串的最後字元不相同，那最後一個字元就不可能同時被選到）

4. Bonus Time Subtask 2~3

- 如果$a_{i-1} = b_{j-1}$，那：
- $dp2(i,j,k) = \max\{dp2(i-1,j-1,k), dp(i-1,j-1,k-1)\} + 2(a_{i-1} == \text{‘a’})$
  （有可能是a_{i-2}, b_{j-2}是對應關係，那多選a_{i-1}, b_{j-1}就只是延續同一段；也有可能a_{i-1}, b_{j-1}是新的一段的開始）
- $dp(i,j,k) = \max\{dp(i,j-1,k), dp(i-1,j,k), dp2(i,j,k)\}$
  （有可能a_{i-1}, b_{j-1}並非對應關係；也有可能是對應關係）
- 最終答案就是$dp(N,M,K)$

4. Bonus Time Subtask 2~3

- 依照這個遞迴式，就可以用DP算出答案了！
- $O(NMK)$，MLE 87/90

- 這裡展示兩種DP的方法：迴圈版和遞迴版

4. Bonus Time Subtask 2~3

- 迴圈版

```
int dp[N+1][M+1][K+1], dp2[N+1][M+1][K+1];
fill(dp, 0);
for (i = 1; i <= N; i++) {
  for (int j = 1; j <= M; j++) {
    if (a[i-1] == b[j-1]) {
      for (k = 1; k <= K; k++) {
        dp2[i][j][k] = max(dp2[i-1][j-1][k], dp[i-1][j-1][k-1]) + 2 * (a[i-1] == 'a');
        dp[i][j][k] = max(dp[i][j][k], dp2[i][j][k]);
      }
    } else {
      for (k = 1; k <= K; k++) dp[i][j][k] = max(dp[i-1][j][k], dp[i][j-1][k]);
    }
  }
}
print(dp[N][M][K]);
```

4. Bonus Time Subtask 2~3

- 遞迴版

```
int dp[N][M][K], dp2[N][M][K];
void Calculate(i, j, k) {
  if (dp[i][j][k] != -1) return; // 已經算過了
  if (i==0 or j==0 or k==0) {    // base case
    dp[i][j][k] = dp2[i][j][k] = 0;
    return;
  }
  Calculate(i-1, j, k); Calculate(i, j-1, k);
  dp[i][j][k] = max(dp[i-1][j][k], dp[i][j-1][k]);
  dp2[i][j][k] = 0;
  if (a[i-1] == b[j-1]) {
    Calculate(i-1, j-1, k); Calculate(i-1, j-1, k-1);
    dp2[i][j][k] = max(dp2[i-1][j-1][k], dp[i-1][j-1][k-1]) + 2 * (a[i-1] == 'a');
    dp[i][j][k] = max(dp[i][j][k], dp2[i][j][k]);
  }
}
main() {
  fill(dp, -1);
  Calculate(N, M, K);
  print(dp[N][M][K]);
}
```

4. Bonus Time

- 空間$O(NMK)$太大了
- 這題要AC一定要用迴圈版
- 在計算dp/dp2[i][][]的時候只會用到dp/dp2[i][][]和dp/dp2[i-1][][]!
- 因為i從小算到大，所以把dp/dp2[i]算完之後dp/dp2[i-1]就不會再被用了，空間可以重複利用
  - 開兩列的空間就好了
- 空間複雜度變成$O(MK)$，AC 100

4. Bonus Time

- 其實寫起來幾乎一樣

```
int dp[2][M+1][K+1], dp2[2][M+1][K+1];
fill(dp, 0); fill(dp2, 0);
for (ti = 1, i = 1; ti <= N; ti++, i ^= 1) {
  fill(dp[i], 0), fill(dp2[i], 0);
  for (int j = 1; j <= M; j++) {
    if (a[ti-1] == b[j-1]) {
      for (k = 1; k <= K; k++) {
        dp2[i][j][k] = max(dp2[i^1][j-1][k], dp[i^1][j-1][k-1]) + 2 * (a[ti-1] == 'a');
        dp[i][j][k] = max(dp[i][j][k], dp2[i][j][k]);
      }
    } else {
      for (k = 1; k <= K; k++) dp[i][j][k] = max(dp[i^1][j][k], dp[i][j-1][k]);
    }
  }
}
print(dp[N&1][M][K]);
```

5. Minecraft Subtask 1~5

- 這題和HW5-2其實很像
- 以下把「背包裡面每一格和自己手上的鑽石數量的組合」簡稱「狀態」
- 假設每一種可能的狀態都是一個節點
- 如果從一種狀態可以點一次滑鼠走到另一種狀態，就在這兩個狀態對應的節點之間畫一條有向邊
- 原問題就變成了最短路徑問題!
  - 因為每條邊都是點一次滑鼠，相當於每條邊邊權相同

5. Minecraft Subtask 1~5

- 既然是無邊權的最短路徑問題，那就BFS吧!

- 要怎麼存一個節點有沒有被走過、和它離出發點的距離呢?
- 開一個map或unordered_map就可以了（key是狀態、value是距離）
- 說起來簡單，但實作上比較複雜

5. Minecraft Subtask 1~5

- 當BFS走到某一個節點，接下來要找的是它可以走到哪些狀態
- 枚舉背包裡面每一格，然後看看在那一格上面按左鍵或右鍵分別會走到哪個狀態
- 實際上只要枚舉每一格有東西的和一格空的就可以了

5. Minecraft Subtask 1~5

- 要怎麼判斷兩個狀態相不相等?
- 首先，假如知道背包內容，因為總鑽石數量固定，所以手上拿幾個其實不需要存
- 第二，因為背包內容是無序的，例如(3,5,1)等價於(1,5,3)，所以判斷相等之前要先sort再判斷
  - 實作上的話，一種寫法是狀態用multiset存（因為可能有幾個格子數量相同）；一種是用vector存，然後每次產生新的狀態之後都要馬上sort
- C++的vector和multiset都有內建operator<, operator==和hash，所以都可以直接塞進map或unordered_map

5. Minecraft Subtask 1~5

- TLE 96

```
unordered_map<set<int>, int> dis;
queue<set<int>> queue;
void visit(state, len) { // 造訪一個狀態
  if (N + M - sum(now) == K) { print(len + 1); exit; }
  // 還沒走過就放進queue
  if (dis.insert({state, len + 1}).second) queue.push(state);
}
void insert_visit(state, num, len) {
  // 造訪當前新增一格包含num個鑽石的狀態
  state.insert(num);
  visit(state, len);
  state.erase(state.find(num));
}

queue.push({N});
while (1) {
  set<int> now = queue.front(), next = now;
  queue.pop();
  int hold = N + M - sum(now), len = dis[now];
  // 按在空的格子上
  if (hold > 0) insert_visit(next, hold, len), insert_visit(next, 1, len);
  for (i in now) { // 按在有鑽石的格子上
    next.erase(next.find(i));
    if (hold > 0) insert_visit(next, i + 1, len), insert_visit(next, i + hold, len);
    else {
      visit(next, len);
      if (i > 1) insert_visit(next, i / 2, len);
    }
    next.insert(i);
  }
}
```

5. Minecraft

- 最後一個Subtask其實是要把判斷兩個集合相不相等寫得快一點
- 最好的方法是像HW3一樣，自己寫個hash函數
- 如果兩個集合的hash值相同，就當這兩個集合相同（賭它不會碰撞）
- 集合的hash要怎麼寫呢?

5. Minecraft

- 首先建立一個隨機數構成的陣列arr[]
- 如果背包有一格包含x個鑽石，就把hash值加上arr[x]
- 因為hash值是加起來的，所以hash出來的結果和順序無關，狀態用vector存就可以了！（省去了sort或multiset的時間）
- 注意arr[]中的隨機數要long long範圍才不容易碰撞
  - rand()產生出來的隨機數是int範圍，要特別注意
- AC

6. Peipei and Frenchfries II

- 提示：二分搜

7. 中國隊列問題

- 提示：linked list

8. 逆序數對

- 提示：Merge sort（或直接google題目名稱也行）

9. 死對頭問題

- 提示：John的走法一定是先走A到x的最短路徑，然後走x->y這條單行道，然後再走y->B的最短路徑。

10. LCS

- 提示：100分是上課內容，不會的請重新複習投影片（拿100分的話陣列開2000就好，不然可能連100分都拿不到）。

  最後20分是要把空間壓到$O(N)$，有興趣的歡迎挑戰（和上課內容不太有關係，所以這裡也就不給提示了）

11. 旅行社大特價

- 提示1：DP。狀態：$dp(i,j)$代表從某一個特定點出發經過恰好i個行程後到達j至少需要花多少錢。
  對於一個固定的i，所有的$dp(i,j)$應該要可以在$O(N^2)$算出來，也就是總複雜度$O(N^3)$。
- 提示2：那個特定點要選哪個點呢？能不能對原本的圖做一點點變化呢？可以先想想看，算完所有的$dp(i,j)$之後，要怎麼知道答案？