練習題6~11題解

附虛擬碼

6. Peipei and Frenchfries II Subtask 1~5

- 這題其實是「最長（非嚴格）遞減子序列」的長度，就是跟HW5-1完全一樣!
  - 只是把遞增改成遞減，如果懶得改的話直接把輸入取個負號就好了(?
- 首先，很明顯的是答案一定大於等於最長遞減子序列長度，因為按照規則，沒有一個人可以吃到子序列上兩天的大薯
- 比較難證的是證明一定可以用剛好等於最長遞減子序列長度的人數把大薯吃完──這個留待下個subtask說明
- 總之$O(N^2)$，TLE/MLE 50（因為只有要求長度，所以空間可以用練習題4的方法壓到$O(N)$）

6. Peipei and Frenchfries II

- 簡單的想法：不如來模擬大薯被買完的過程如何?
- 維護一個可變長度陣列D[]，代表現在有買過大薯的人，每個人最後一次買大薯的份量是多少
- 如果接下來一天的大薯份量小於等於所有人最後一次買大薯的份量，那勢必得多一個人來買→增加一個人
- 如果大於，就不需要多一個人買→挑一個可以買這份薯條的人，更新他最後一次買的份量

6. Peipei and Frenchfries II

- 那如果有很多個可以買這份薯條的人，要挑誰呢?
- 一個直覺的想法是挑最後一次買的份量最多的那個人
- 事實上是對的（最佳策略），可以用數學歸納法證明，但寫起來有一點複雜，在這裡就不耗費篇幅證明

6. Peipei and Frenchfries II

- 注意到如果按照這個更新方法的話，D[]一定會是遞減的!
- 假設原本的D[]遞減，當天份量x，則:
- 「最後一次買的份量最多且可以買這份薯條的人」i，代表比他多的人（也就是前一個元素）沒辦法買這份薯條（$\mathrm{D}[i-1] \geq x$）；也就是說，把$\mathrm{D}[i]$改成x，原數列仍然遞減
  - 注意如果有很多個一樣大的元素，那要挑最前面那個更新
- x小於等於D[]中所有的元素，則新的那個人被push_back後最後一天買的份量就是x，因此仍然保持遞減

6. Peipei and Frenchfries II

- 既然遞減，那用二分搜就可以找到要更新誰了！
- $O(N \log N)$，AC

```
vector<int> D;
for (i in a) {
  iterator it = upper_bound(D.begin(), D.end(), -i);
  if (it == D.end()) D.push_back(-i);
  else *it = -i;
}
print(D.size());
```

- 如果把D[i]看成「到目前為止，長度為i的所有遞減子序列中，結尾最大可以是多少」，就可以發現這和最長遞減子序列是同樣的問題了！

7. 中國隊列問題

- 這題其實最好寫的就是正解，所以就直接講正解怎麼寫(?
- 相信應該不難看出這題只要寫一個doubly linked list就可以了

7. 中國隊列問題

- 基本操作：連結兩個節點A, B（A後=B; B前=A;）
- 在X, Y間插入一個節點A：連結(X, A)和(A, Y)
- 重組(A, B, C)：連結(A前, B後)、(C前, A)和(B, C)
  - 注意操作順序，這裡指的前後都是重組前的
- 刪除節點A：連結(A前, A後)
- 刪除操作就一個一個刪，刪到發完或到末端就停
  - 記得要記頭尾是誰，這樣刪除的時候才能知道甚麼時候要停
- 千萬不要空想，拿張紙出來畫會快很多！

7. 中國隊列問題

- 插入和重組都是$O(1)$（要存好每個人節點的指標），又因為每個人只會離開隊列一次，所以所有刪除加起來是$O(N)$
- 總複雜度$O(N+Q)$，AC

- 這裡講一些實作細節
- 要特判頭尾，還要小心不要存取到nullptr?
- 一個簡單的做法是新增兩個「虛擬」的人，一個在頭一個在尾，然後刪除的時候遇到這兩個人就停下來，就不用特判了

7. 中國隊列問題

- 指標好難寫，還要new/delete?
- 不需要那麼麻煩!
- 實際上doubly linked list的意思就是紀錄每個人前後是誰
- 那就開兩個陣列prv[]和nxt[]，prv[i]代表p_i人的前一個人編號是多少，以此類推
- 這樣可以大幅減少出bug的機率(?

7. 中國隊列問題

```
int nxt[N+3], prv[N+3];
function Conn(a, b) { nxt[a] = b; prv[b] = a; }
Conn(N + 1, a0); Conn(a0, N + 2); // head & tail
for ((k, a, b, c) in query) {
  if (k == 1) {
    if (c == 1) {
      Conn(prv[b], a); Conn(a, b);
    } else {
      Conn(a, nxt[b]); Conn(b, a);
    }
  } else if (k == 2) {
    Conn(prv[a], nxt[b]);
    Conn(prv[c], a);
    Conn(b, c);
  } else {
    int i, j;
    if (c == 1) {
      for (i = a, j = nxt[a]; i <= N && b; i = prv[i], b--) ans.push_back(i);
    } else {
      for (j = a, i = prv[a]; j <= N && b; j = nxt[j], b--) ans.push_back(j);
    }
    Conn(i, j); cnt += b;
  }
}
print(cnt);
for (i in ans) print(i);
```

8. 逆序數對 Subtask 1~5

- 枚舉所有的pair，暴力$O(N^2)$!
- TLE 35

- Subtask 2的話只有兩種數字，只要把「每個2後面有幾個1」全部加起來就好了
- 可以用$O(N)$預處理a_i~a_{N-1}有幾個1，然後再花$O(N)$加起來
- TLE 50

8. 逆序數對 Subtask 6

- 考慮Insertion sort，可以發現每交換一次就是少一個逆序數對
  - 排序好之後就沒有逆序數對
- 數Insertion sort交換了幾次就好（或者是往前移了幾格的總數）
  - 注意等於就不要再交換了
- 事實上Insertion sort的複雜度就是$O(N+K)$，K是逆序數對總數
- TLE 60（加上Subtask 2）

8. 逆序數對

- 不如來試試看$O(N \log N)$的排序?
- 考慮Merge sort
- 在合併的時候，假設左右兩邊各自的逆序數對都已經被算到了
- 還沒算到的是橫跨兩邊的逆序數對
- 當左邊的某個數a大於右邊某個數b的時候，代表左半邊在a右邊（包含a本身）的所有數原本都大於b，因此把答案加上這個數量
- 這樣遞迴下去就可以算到所有逆序數對了
- AC 100
- 之這題的輸入測資其實跟Practice 6完全一樣XD

8. 逆序數對

```
int a[N], b[N];
ans = 0;
function Mergesort(l, r) { // [l, r)
  if (r - l == 1) return;
  int m = (l + r) / 2;
  Mergesort(l, m); Mergesort(m, r);
  copy(a + l, a + r, b + l);
  int i = l, j = m, k = l;
  for (; i < m && j < e; k++) {
    if (b[j] < b[i]) {
      a[k] = b[j++]; ans += m - i;
    } else {
      a[k] = b[i++];
    }
  }
  copy(b + i, b + m, a + k);
  copy(b + j, b + r, a + k);
}
Mergesort(0, N);
print(ans);
```

9. 死對頭問題 Subtask 2

- 只有唯一的路線可以通行，代表一定要回頭走一次
- 事實上這樣子圖就會是一棵樹（每條邊都是雙向）
- 回頭走的那條邊會來回各走一次
- 枚舉路徑上每一條邊（這可以DFS的時候順便做），然後找到那條邊的過路費來回加起來最小的就可以了
- WA 30
- 考試的時候這種簡單的測資記得要拿(?

9.死對頭問題 Dijkstra's Algorithm Revised

- 上課有講過Dijkstra's algorithm，但是沒有在上課時說明複雜度$O((N+M)\log M)$的作法，所以在這裡補充
  - 事實上教授的投影片（http://mirlab.org/jang/courses/dsa/slide/13.5-shortestpath.ppt，16~17頁）也有
- Dijkstra每一次是挑當前不在最短路徑樹上，且離原點最近的點加入最短路徑樹，並用它來鬆弛（relax）其他點
- 如果暴力找的話每次找就要花$O(N)$
- 但是看起來只是要每次找到距離最小的點、並在鬆弛的時候更新距離，說不定可以用heap?

9.死對頭問題 Dijkstra's Algorithm Revised

- 事實上是可以的，而且可以用priority_queue達成!
- priority_queue裡面放(離起點距離, 點)的pair，並讓top是離起點距離最近的
  - priority_queue預設是max heap，所以要取個負號或自己寫比較運算
- 重點是如何更新點權，因為priority_queue沒有辦法刪除或修改一個特定的值

9.死對頭問題 Dijkstra's Algorithm Revised

- 注意到對點進行鬆弛後該點離起點距離一定會變小
- 所以就算不把priority_queue裡面那點拔出來也無所謂，因為第一次從priority_queue拿到那點一定是正確的值（因為是min heap）
- 開一個bool陣列紀錄每個點是不是已經在最短路徑樹中了，這樣子第二次以後拿到同一個點的時候，看他已經在最短路徑樹中，就可以知道這是已經過期的值，可以直接略過

9.死對頭問題 Dijkstra's Algorithm Revised

- 寫起來大概像這樣
- priority_queue是用operator<，所以自己定義這個函數就可以直接讓它變成min heap

```
struct Edge {
  int pt; long long dis;
  bool operator<(const Edge& i) const {
    return dis > i.dis; // comparison function
  }
};
bool in_tree[N];
function Dijkstra(start, dist, edges) {
  // dist: distance from start, edges: adjacency list (array of vector<Edge>)
  priority_queue<Edge> pq;
  fill(in_tree, false); fill(dist, Infinity);
  dist[start] = 0;
  pq.push((Edge){start, 0});
  while (1) {
    while (!pq.empty() && in_tree[pq.top().pt]) pq.pop();
    if (pq.empty()) return;
    now = pq.top().pt;
    in_tree[now] = true;
    for (i in edges[now])
      if (!in_tree[i.pt] && dist[i.pt] > dist[now] + i.dis)
        pq.push((Edge){i.pt, dist[i.pt] = dist[now] + i.dis}); // relax
  }
}
```

9. 死對頭問題 Subtask 2~4

- 注意到John一定會走A⇝X的最短路徑，然後走一條邊到Y，再走最短路徑到B
- 因為如果X→Y走超過一條邊的話，那他必定不會是對John最好的路徑；或者是其實是，但是在X⇝Y中存在一條邊X'→Y'，使得走A⇝X'→Y' ⇝B和走A⇝X⇝Y⇝B花的錢相同

- 枚舉每一條邊，然後分別求兩邊的最短路徑
  - 記得要把剛好等於最短路徑長度的路徑排除掉
- $O(M^2 \log M)$或$O(MN \log M)$，TLE 50（加上Subtask 2）

9. 死對頭問題

- 剛才那個演算法主要卡在對每個點Y都要求Y⇝B的最短路徑
- 但是Y⇝B在原圖上的最短路徑，等於把圖所有邊都反過來之後B⇝Y的最短路徑
- 所以其實先求兩次最短路徑樹，就可以對每條邊$O(1)$算出來了！（原圖上A⇝all，和反過來的圖上B⇝all）
- $O((N+M)\log M)$，AC

9. 死對頭問題

- Dijkstra寫好就一下就寫完了

```
vector<Edge> graph[N], reversed[N];
long long dis1[N], dis2[N];
Dijkstra(A, dis1, graph);
Dijkstra(B, dis2, reversed);
ans = Infinity;
for ((i, j, w) in all edges) { // (from, to, weight)
  len = dis1[i] + dis2[j] + w;
  if (len != dis1[B] && len < ans) ans = len;
}
print(ans - dis1[B]);
```

10. 最長共同子序列 Subtask 2~5

- 拿不到100分請回去看上課投影片
  - 陣列開2000就好

```
int dp[N+1][M+1];
char ans[];
fill(dp, 0); fill(ans, 0);
for (i = 0; i < N; i++)
  for (j = 0; j < M; j++)
    dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j], a[i] == b[j] ? dp[i][j] + 1 : 0);
for (i = N, j = M; dp[i][j];) {
  if (dp[i][j-1] == dp[i][j]) j--;
  else if (dp[i-1][j] == dp[i][j]) i--;
  else {
    i--, j--;
    ans[dp[i][j]] = a[i];
  }
}
print(ans);
```

10. 最長共同子序列 Subtask 6

- 這部分跟考試範圍不太有關，不想看的可以自行跳到下一題(?
- 空間太少了，$O(N^2)$開不下！但是$O(N)$只能算長度，不能把LCS存起來
- 那能不能一次算一小部分就好?
- 考慮只把最後的$K+1$排dp[][]存起來（dp[N-K][]~dp[N][]）
- 這樣可以知道LCS最後的部分（包含A[k]~A[N-1]的部分），還可以順便知道他對應到B字串的哪個部分!

10. 最長共同子序列 Subtask 6

- 假設最後面這部分的LCS是由A[K]~A[N-1]和B[j]~B[M-1]構成的，那就代表前面那些部分的LCS是由A[0]~A[K-1]和B[0]~B[j-1]構成的
- 重複做N/K輪就可以把整個LCS求出來了！
- 時間複雜度 $O(N^3/K)$，空間複雜度$O(KN)$
- K取大概80可以過Subtask 6

10. 最長共同子序列

- 整題要求時間複雜度 $O(N^2)$，空間複雜度$O(N)$
- 考慮採用只求長度的DP，但是在DP陣列中多記錄一個值i，代表他是從dp[N/2][i]轉移來的
- 於是我們在做完之後，看dp[N][M]的i值可以知道LCS是由（A[0]~A[N/2-1]和B[0]~B[i-1]的LCS）和（A[N/2]~A[N-1]和B[i]~B[M-1]的LCS）組合成的
- 兩邊分別遞迴下去，用一樣的方法求，到某一個字串長度只剩下1就可以直接求出答案

10. 最長共同子序列

- 注意到第一層跑完一次DP的時間是$O(N^2)$
- 兩個第二層加起來各跑完DP則會是第一層的一半（因為遞迴下去的兩個A字串長度變成一半）
- 這樣下去，會得到$O\left(N^2+\frac{N^2}{2}+\frac{N^2}{4}+\cdots\right)=O(N^2)$
- 而用到的空間只有$O(N)$
- AC

11. 旅行社大特價

- 這題其實也是個DP
- DP狀態：dp[k][i]代表從起點走恰好k步後結束在點i的最短路徑
- 起點是甚麼？從題目中其實可以看出來，就是自己新增一個點，然後用免費（邊權0）的邊連到所有點
  - 解決了圖可能不連通或不強連通（有些點走不到另外一些點）的問題

- 一樣，可以先試著停在這裡寫寫看遞迴式

11. 旅行社大特價

- $dp[k][i] = \min_{(j,i) \text{ in edges}}(dp[k-1][j] + A[j][i])$
- 很好理解，就只是枚舉走到點i前一步是在哪裡

- 重點是這個dp到底有甚麼用?
- 注意到本題要求的是求出一個環使得平均權重最小
- 這裡來觀察兩個重要的性質

11. 旅行社大特價

- (1) 如果圖上沒有負環，但是存在一個零環（一個環使得環上所有邊權加起來等於零），那$\min_i \max_k \frac{dp[N+1][i]-dp[k][i]}{N+1-k} = 0$
- 因為沒有負環，所以$\max_k \frac{dp[N+1][i]-dp[k][i]}{N+1-k} \geq 0$，但是存在一個零環上的點$x$使得$\max_k \frac{dp[N+1][x]-dp[k][x]}{N+1-k} = 0$
- (2) 如果把所有的邊權減少v，則最小平均值環的位置不改變（當然平均值會下降v），同時$\frac{dp[N+1][i]-dp[k][i]}{N+1-k}$也會下降$v$，並且在$v \leq$ 最小平均值環的平均 時，不會產生負環

11. 旅行社大特價

- 由這兩個性質可以推得$\min_i \max_k \frac{dp[N+1][i]-dp[k][i]}{N+1-k}$就是最小平均值環的平均值!
- 算DP用$O(N^3)$，算出最後這個值會花$O(N^2)$，總複雜度$O(N^3)$
- 看起來有一點點大，但是因為這個DP算起來非常簡單，因此其實是會AC的

11. 旅行社大特價

```
long long dp[N+1][N+1];
long long A[N+1][N+1]; // input is A[1~N][1~N]
for (all (i, j) such that j=0 or A[i][j]=0) A[i][j] = Infinity;
fill(dp, Infinity);
for (i = 1; i <= N; i++) dp[0][i] = 0;
for (k = 1; k <= N; k++)
  for (i = 0; i <= N; i++)
    for (j = 0; j <= N; j++) dp[k][i] = min(dp[k][i], dp[k - 1][j] + A[j][i]);
Fraction ans = Infinity;
for (i = 1; i <= N; i++) {
  if (dp[N][i] == Infinity) continue;
  Fraction now = -Infinity;
  for (j = 0; j < N; j++) now = max(now, (Fraction){dp[N][i] - dp[j][i], N - j});
  ans = min(ans, now);
}
if (ans == Infinity) print("-1 -1");
else print(ans);
```

11. 旅行社大特價

- 有理數的部分理論上long double精度應該是夠的，不過想要自己寫有理數也不難

```
long long gcd(long long a, long long b) {
  while (a) { b %= a; std::swap(a, b); }
  return b;
}
struct Fraction {
  long long a, b;
  bool operator<(const Fraction& x) const {
    return a * x.b < b * x.a;
  }
  void Reduce() { long long tmp = gcd(a, b); a /= tmp, b /= tmp; }
};
```