基础解法

Java代码

```
class Solution {
    public int lenLongestFibSubseq(int[] A) {
        int N = A.length;
        Set<Integer> S = new HashSet();// 使用set来存储便于快速查找
        for (int x: A) S.add(x);•         int res = 0;
        for (int i = 0; i < N; ++i)
            for (int j = i+1; j < N; ++j) {
                int x = A[j]; // 假设从pair(A[i], A[j]) 开始
                int y = A[i] + A[j];// y 代表 A[i]->A[j] 的下一个斐波那契数列的元素
                int length = 2;
                while (S.contains(y)) { // 继续向后搜索以pair (A[i], A[j])开始的斐波那契数列
                    // x, y -> y, x+y
                    int tmp = y;
                    y += x;
                    x = tmp;
                    res = Math.max(res, ++length);
                }
            }
•         return res >= 3 ? res : 0;
    }
}
```

最优解法

Java代码

```
class Solution {
    public int lenLongestFibSubseq(int[] A) {
        int N = A.length;
        Map<Integer, Integer> index = new HashMap();// 建立值到下标的映射便于快速查找
        for (int i = 0; i < N; ++i)
            index.put(A[i], i);

        Map<Integer, Integer> longest = new HashMap();
```

```
        int ans = 0;

        for (int k = 0; k < N; ++k)
            for (int j = 0; j < k; ++j) {
                int i = index.getOrDefault(A[k] - A[j], -1);
                if (i >= 0 && i < j) {
                    // 把(i, j) 对应的位置编码成 (i * N + j)，这样就可以用一维数组来存储状态
                    int cand = longest.getOrDefault(i * N + j, 2) + 1; // 利用状态转移方程来求解
                    longest.put(j * N + k, cand);// 记录中间状态值，便于后续求解时调用
                    ans = Math.max(ans, cand);
                }
            }

        return ans >= 3 ? ans : 0;
    }
}
```

C++代码

```
class Solution {
public:
    int lenLongestFibSubseq(vector<int>& A) {
        int N = A.size();
        unordered_map<int, int> index;
        for (int i = 0; i < N; ++i)
            index[A[i]] = i;

        unordered_map<int, int> longest;
        int ans = 0;
        for (int k = 0; k < N; ++k)
            for (int j = 0; j < k; ++j) {
                if (A[k] - A[j] < A[j] && index.count(A[k] - A[j])) {
                    int i = index[A[k] - A[j]];
                    longest[j * N + k] = longest[i * N + j] + 1;
                    ans = max(ans, longest[j * N + k] + 2);
                }
            }

        return ans >= 3 ? ans : 0;
    }
};
```

Python代码

```
class Solution(object):
    def lenLongestFibSubseq(self, A):
        index = {x: i for i, x in enumerate(A)}
        longest = collections.defaultdict(lambda: 2)

        ans = 0
        for k, z in enumerate(A):
            for j in xrange(k):
                i = index.get(z - A[j], None)
                if i is not None and i < j:
                    cand = longest[j, k] = longest[i, j] + 1
                    ans = max(ans, cand)

        return ans if ans >= 3 else 0
```