Assignment XI - DSAA(H)

Name: Yuxuan HOU (侯宇轩)

Student ID: 12413104

Date: 2025.11.23

Question 11.1 (0.25 marks)

Consider a modification of the rod-cutting problem where in addition to a price p_i for each rod, each cut incurs a fixed cost c. The revenue for a solution is now the sum of the prices of the individual pieces minus the costs of making the cuts. Give a Dynamic Programming Bottom-Up algorithm to solve this problem.

Sol:

Let $R(n)$ represents the answer with length n, then with cut of length i, we have $R(n) = p_i + R(n-i) - c$.

Therefore:

$$R(n) = \max\left(p_n,\ \max_{1 \le i < n}\{p_i + R(n-i) - c\}\right)$$

Pseudo Code:

Algorithm 1: DP(p, n, c)

let $r[0 \dots n]$ be a new array
$r[0] \leftarrow 0$
for $j \leftarrow 1$ **to** n **do**
    $q \leftarrow -\infty$
    for $i \leftarrow 1$ **to** j **do**
        if $i = j$ **then**
            $temp \leftarrow p_j$
        end
        else
            $temp \leftarrow p_i + r[j - i] - c$
        end
        if $temp > q$ **then**
            $q \leftarrow temp$
        end
    end
    $r[j] \leftarrow q$
end
return $r[n]$

Question 11.2 (0.25 marks)

```
MEMOIZED-CUT-ROD(p, n)
let r[0 : n] be a new array          // will remember solution values in r
for i = 0 to n
    r[i] = −∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0                 // already have a solution for length n?
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n          // i is the position of the first cut
        q = max {q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r)}
r[n] = q                    // remember the solution value for length n
return q
```

Modify MEMOIZED-CUT-ROD so that it also returns the actual solution rather than just its value.

Sol:

Algorithm 1: Memoized-Cut-Rod(p, n)

```
let r[0...n] be a new array
let s[0...n] be a new array
for i ← 0 to n do
    r[i] ← −∞
    s[i] ← 0
end
r[n] ← MEMOIZED-CUT-ROD-AUX(p, n, r, s)
return (r[n], s)
```

Algorithm 2: Memoized-Cut-Rod-Aux(p, n, r, s)

```
if r[n] ≥ 0 then
    return r[n]
end
if n = 0 then
    q ← 0
end
else
    q ← −∞
end
for i ← 1 to n do
    temp ← p[i]+ MEMOIZED-CUT-ROD-AUX(p, n − i, r, s)
    if temp > q then
        q ← temp
        s[n] ← i
    end
end
r[n] ← q
return q
```

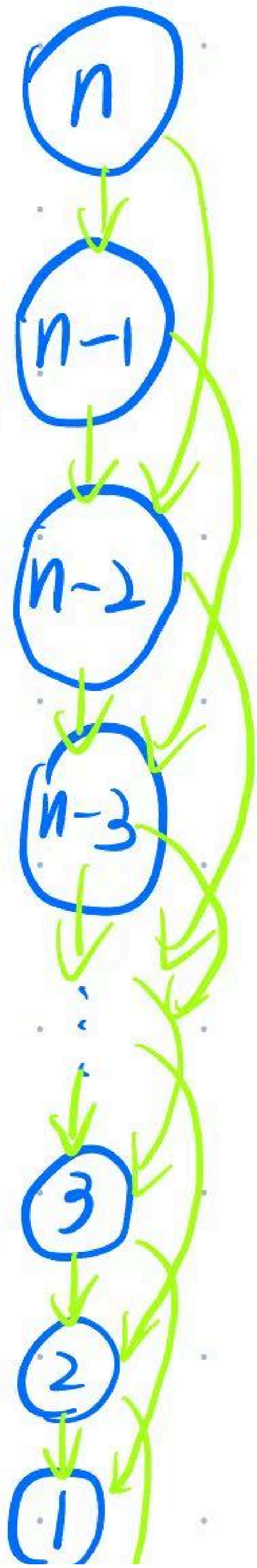
Question 11.3 (0.25 marks)

Provide the pseudo-code of an $O(n)$ time Dynamic Programming algorithm for calculating the n_{th} Fibonacci number. Draw the subproblem graph. How many vertices and edges does the subproblem graph contain?

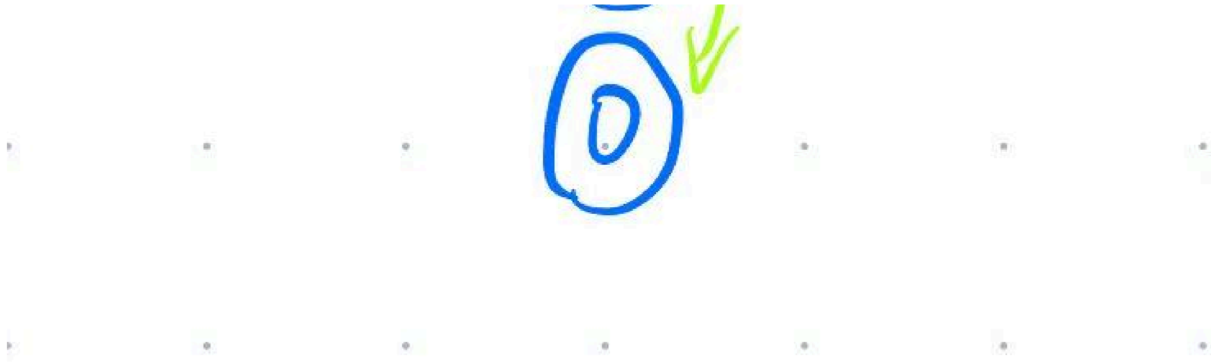
Sol:

Algorithm 1: Fibonacci(n)

if $n = 0$ **then**
    return 0
end
if $n = 1$ **then**
    return 1
end
let $F[0 \ldots n]$ be a new array
$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
for $i \leftarrow 2$ **to** n **do**
    $F[i] \leftarrow F[i-1] + F[i-2]$
end
return $F[n]$



n
n-1
n-2
n-3
3
2
1



Thus we obtain that the number of nodes: $|V| = n + 1$, edges: $|E| = (n - 1) \times 2 = 2n - 2$.

Question 11.4 (1 mark)

After retiring from a large company, Mr Mortimer wonders how much money he could have made if he had invested his life savings in shares of his company. He looks back at the share prices over the n days of his employment and asks himself how much his investment could have returned if he had—somehow—known the best time to buy and sell his shares. Mortimer doesn't like trading shares, so he would only ever buy once and sell once and assume that his life savings is a fixed amount.

Let $a_1, \ldots, a_n$ describe the difference of share prices over time: a_i is the amount of money Mr Mortimer would have gained if he had held on to shares on day i. Note that a_i can be negative, in which case day i would have been a loss. Assume $a_i \neq 0$. Let $f(i, j) = \sum_{k=i}^{j} a_k$ be the money earned if Mortimer had bought shares at the start of day i of his employment and sold them at the end of day j. Mortimer wants to find the maximum return $\max\{f(i, j) \mid 1 \leq i \leq j \leq n\}$.

Example: for $a_1, \ldots, a_n = +5, -3, -4, +8, -1, +12, -6, +4, +4, -14, +2, +8$, Mortimer's maximum return would have been $f(4, 9) = 8 + (-1) + 12 + (-6) + 4 + 4 = 21$ pounds.

(a) Design an algorithm in pseudocode that computes the maximum return using dynamic programming in time $O(n)$, following the hints below. Explain your solution.

(b) Show that your algorithm runs in time $O(n)$.

Hints: Use a bottom-up approach, tabulating for each day k:

- the maximum return A_k for an investment up to day k (buying and selling up to day k, formally $\max\{f(i, j) \mid 1 \leq i \leq j \leq k\}$) and
- the maximum return B_k up to day k for an *ongoing* investment: still holding on to shares on day k (formally $\max\{f(i, k) \mid 1 \leq i \leq k\}$).

Work out how A_1 and B_1 can be initialised, and work out Bellman equations showing how, for $k \geq 2$, A_k and B_k can be computed based on the input and values of A and B that you have tabulated earlier. Pay attention to the order in which to tabulate values. Don't forget to state how to compute the final output from the tabulated values.

Sol:

1.

Algorithm 1: MaxReturn($a_1, \ldots, a_n$)

```
let A[1...n] and B[1...n] be new arrays
B[1] ← a_1
A[1] ← max{0, a_1}
for k ← 2 to n do
    B[k] ← max{B[k − 1] + a_k, a_k}
    A[k] ← max{A[k − 1], B[k]}
end
return A[n]
```

As the pseudo code goes, $B[i]$ represents the best interval ends with i, whose two options are connect with $B[i-1]$ or start with new one. Then the maximum of $B[i]$, which is $A[n]$ will be the eventual answer.

2. As the pseudo code goes, there only contains a for-loop which is obviously linear and some $\Theta(1)$ operations. Therefore, the runtime is $\Theta(n)$, then it must be $O(n)$.

Initialization

On day 1, the $B[1]$ must contains a_1, thus $B[1] = a_1$. For $A[1]$, it could be $B[1]$ or nothing, which is $\max\{B[1],\ 0\}$.

Bellman Equations

1. **Ongoing B_k**

   At day k we either extend the best interval ending at day $k-1$, or start a new interval on day k.

   Thus,

$$B_k = \max\{B_{k-1} + a_k,\ a_k\}$$

2. **Finished A_k**

By day k we either already had the best finished investment on or before day $k-1$, or finish an ongoing investment at day k, which gives value B_k.

Thus,

$$A_k = \max\{A_{k-1},\ B_k\}$$

Final Output

After computing up to day n, the maximum return overall is A_n.

Question 11.5 (0.5 marks)

Implement the three problems "Longest common subsequence", "Stone Merging" and "Task arrangement" on the OJ system.

Sol:

题目

状态	最后递交于	题目
✓ 100 Accepted	4 小时前	**49** Longest Common Subsequence
✓ 100 Accepted	4 小时前	**50** Stone Merging
✓ 100 Accepted	17 分钟前	**51** Task Arrangement

```
int main(){
    string s, t; cin >> s >> t;

    int N((int)s.size()), M((int)t.size());

    vector < vector < int > > dp(N + 10, vector < int >(M + 10, 0));

    for(int i = 1; i <= N; ++i){
        for(int j = 1; j <= M; ++j){
            if(s[i - 1] == t[j - 1])dp[i][j] = dp[i - 1][j - 1] + 1;
```

```
                else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
    }

    int i(N), j(M);
    string ans;
    while(i > 0 && j > 0){
        if(s[i - 1] == t[j - 1])ans.push_back(s[i - 1]), --i, --j;
        else if(dp[i - 1][j] >= dp[i][j - 1])--i;
        else --j;
    }
    reverse(ans.begin(), ans.end());
    cout << ans << '\n';

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}
```

```
int main(){
    int N = read(), M1 = read(), M2 = read();
    vector < int > D1(N + 10, 0), D2(N + 10, 0);
    vector < pair < int, int > > air1, air2;
    for(int i = 1; i <= M1; ++i){
        int s = read(), t = read();
        air1.push_back({s, t});
    }
    for(int i = 1; i <= M2; ++i){
        int s = read(), t = read();
        air2.push_back({s, t});
    }
    sort(air1.begin(), air1.end(), [](const pair < int, int > &a, const pair < int, int > &b)->bool{
        return a.first == b.first ? a.second < b.second : a.first < b.first;
    });
    sort(air2.begin(), air2.end(), [](const pair < int, int > &a, const pair < int, int > &b)->bool{
```

```
        return a.first == b.first ? a.second < b.second : a.first < b.first;
    });
    auto cmp = [](const pair < int, int > &a, const pair < int, int > &b)->bool{
        return a.first == b.first ? a.second > b.second : a.first > b.first;
    };
    priority_queue < pair < int, int >, vector < pair < int, int > >, decltype(cmp) > cur(cmp);
    priority_queue < int, vector < int >, greater < int > > fre;
    int lft(0);
    for(auto [s, t] : air1){
        while(!cur.empty() && cur.top().first < s)
            fre.push(cur.top().second), cur.pop();
        int idx(-1);
        if(!fre.empty())idx = fre.top(), fre.pop();
        else idx = ++lft;
        if(idx <= N)++D1[idx];
        cur.push({t, idx});
    }
    while(!cur.empty())cur.pop();
    while(!fre.empty())fre.pop();
    lft = 0;
    for(auto [s, t] : air2){
        while(!cur.empty() && cur.top().first < s)
            fre.push(cur.top().second), cur.pop();
        int idx(-1);
        if(!fre.empty())idx = fre.top(), fre.pop();
        else idx = ++lft;
        if(idx <= N)++D2[idx];
        cur.push({t, idx});
    }

    int res(0);
    for(int i = 1; i <= N; ++i)D1[i] += D1[i - 1], D2[i] += D2[i - 1];
    for(int i = 0; i <= N; ++i)res = max(res, D1[i] + D2[N - i]);
    printf("%d\n", res);


```

```
    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}
```

```
int main(){
    int N = read();
    vector < ll > A((N << 1) + 10, 0);
    for(int i = 1; i <= N; ++i)A[i] = A[i + N] = read();

    vector < ll > pre((N << 1) + 10, 0);
    for(int i = 1; i <= N << 1; ++i)pre[i] = pre[i - 1] + A[i];

    const ll INF = LONG_LONG_MAX >> 2;

    vector < vector < ll > > dpMn(410, vector < ll >(410)), dpMx(410, vector < ll >(410));

    for(int len = 2; len <= N; ++len){
        for(int i = 1; i + len - 1 <= N << 1; ++i){
            int j = i + len - 1;
            dpMn[i][j] = INF;
            dpMx[i][j] = 0;
            ll w(pre[j] - pre[i - 1]);
            for(int k = i; k < j; ++k)
                dpMn[i][j] = min(dpMn[i][k] + dpMn[k + 1][j] + w, dpMn[i][j]),
                dpMx[i][j] = max(dpMx[i][k] + dpMx[k + 1][j] + w, dpMx[i][j]);
        }
    }

    ll ansMn(INF), ansMx(0);
    for(int i = 1; i <= N; ++i)
        ansMn = min(ansMn, dpMn[i][i + N - 1]),
        ansMx = max(ansMx, dpMx[i][i + N - 1]);

    printf("%lld\n%lld\n", ansMn, ansMx);

```

```
    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}

```

```
int main(){
    int N = read();
    ll s = read < ll >();

    vector < ll > sumT(N + 10, 0);
    vector < ll > sumC(N + 10, 0);

    for(int i = 1; i <= N; ++i){
        int t = read(), c = read();
        sumT[i] = sumT[i - 1] + (ll)t;
        sumC[i] = sumC[i - 1] + (ll)c;
    }

    const ll INF(0x3f3f3f3f3f3f3f3fLL);
    vector < ll > dp(N + 10, 0);
    for(int i = 1; i <= N; ++i)dp[i] = INF;
    dp[0] = 0;

    ll totC(sumC[N]);

    for(int i = 1; i <= N; ++i){
        ll curT(sumT[i]);
        ll curC(sumC[i]);
        for(int j = 0; j < i; ++j)
            dp[i] = min(dp[i], dp[j] + curT * (curC - sumC[j]) + s * (totC - sumC[j]));
    }

    printf("%lld\n", dp[N]);

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
```

```
}

```