งานสังสรรค์ (Carnival)

Problem by neonaht

โจทย์ข้อนี้ให้หาว่าถ้าเล่นเกมด้วยวิธีที่ดีที่สุดจะได้รับเงินสูงสุดเท่าไหร่ ?

Subtask #1-2 : $N \leq 300$ และ $N \leq 1000$

สำหรับปัญหาย่อยนี้เราสามารถมองการเล่นเกมแต่ละตาเป็นรูปของ DP 2 มิติได้ โดยจะนิยามให้ $dp[i][j]$ คือ จำนวนเงินที่สามารถเก็บได้มากที่สุดเมื่อเล่นจนถึงตาที่ i โดยในตานี้เราเลือกโกยถุงเงินที่ถุงที่อยู่ริมซ้ายสุดคือถุงที่ j และริมขวาสุดคือถุงที่ $i+j-1$ นั่นเอง เมื่อได้นิยามมาแล้วเราจะสามารถเขียนสมการ DP ได้โดยจะให้
max_{out} คือ คำตอบที่ดีที่สุดของตาที่แล้วโดยเลือกให้ไม่มีถุงซ้ำกับตานี้
max_{in} คือ คำตอบที่ดีที่สุดของตาที่แล้วโดยเลือกให้มีถุงซ้ำกับตานี้
val คือ จำนวนเงินที่ได้จากตานี้ (ใช้ Quick Sum เก็บค่าเงินเพื่อความเร็วในการคำนวณ)
สมการ: $dp[i][j] = max(max_{out} + val, max_{in} + \frac{val}{2})$ ซึ่งเราสามารถใช้ for loop ในการหาได้ค่า max ได้

```
const int Size=5007;
long long token[Size],qs[Size],dp[Size][Size];
// Subtask#1 use int
// Subtask#2 use long long

void process(void) {
    int N;
    cin >> N;
    for(int i=1;i<=N;i++) cin >> token[i];
    for(int i=1;i<=N;i++) qs[i]=qs[i-1]+token[i];

    for(int i=1;i<=N;i++) dp[1][i]=token[i];
    long long ans=0;

    for(int len=2;len<=N;len++) {
        int lim=N-len+1;

        for(int l=1;l<=lim;l++) {
            int r=l+len-1;

            for(int i=max(1,l-len+2);i<=r;i++) dp[len][l]=max(dp[len]
[l],dp[len-1][i]+(qs[r]-qs[l-1])/2);

            for(int i=1;i<=l-len+1;i++) dp[len][l]=max(dp[len][l],dp[
len-1][i]+(qs[r]-qs[l-1]));
            for(int i=r+1;i<=N;i++) dp[len][l]=max(dp[len][l],dp[len-
1][i]+(qs[r]-qs[l-1]));

            ans=max(ans,dp[len][l]);
        }
    }
    cout << ans;
    return ;
}
```

Time Complexity: $\mathcal{O}(N^3)$

Subtask #3 : ทุกถุงมีเงินอยู่เท่ากันหมด

เราสามารถ Observe ได้ว่าวิธีเล่นที่ดีที่สุดจะต้องพยายามเลือกให้ไม่ซ้ำมากที่สุด นั่นคือพยายามให้โดนหักเงินน้อยที่สุดซึ่งสามารถทำได้โดยเลือกถุงริมซ้ายสุดกับริมขวาสุดสลับกันไปเรื่อยๆในแต่ละตา จนเมื่อถึงตาที่ $\frac{N+1}{2}+1$ จึงจะมีการเลือกซ้ำเกิดขึ้นครั้งแรกและเป็นไปเรื่อยๆจนถึงตาที่ N ซึ่งจะทำให้ได้รับเงินมากที่สุด

```
const int Size=5007;
long long token[Size],qs[Size];

void process(void) {
    int N;
    long long ans=0;
    cin >> N;
    for(int i=1;i<=N;i++) cin >> token[i];
    for(int i=1;i<=N;i++) qs[i]=qs[i-1]+token[i];

    int observe = (N+3)/2;

    for(int i=1;i<observe;i++) ans+=qs[i];
    for(int i=observe;i<=N;i++) ans+=qs[i]/2;

    cout << ans;
    return ;
}
```

Time Complexity: $\mathcal{O}(N)$

Subtask #4 : วิธีเล่นที่ดีที่สุดในตานี้จะต้องมีถุงที่ซ้ำกับตาที่แล้ว

ในกรณีนี้เราสามารถนำ Solution จาก Subtask #1 มาประยุกต์ใช้โดยเราจะพิจารณาแค่ max_{in} ซึ่งนำ Sliding Window Algorithm มาช่วยล่นเวลาในการทำงานได้ (สามารถเพิ่มเทคนิค State Space มาช่วยลดการใช้เม็มได้)

```
const int Size=5007;
long long token[Size],qs[Size],dp[2][Size];

void process(void) {
    int N;
    cin >> N;

    assert(N<=5000);

    for(int i=1;i<=N;i++) cin >> token[i];
    for(int i=1;i<=N;i++) qs[i]=qs[i-1]+token[i];

    int now=1,last=0;
    deque <pair<long long,int>> Q_in;

    for(int i=1;i<=N;i++) dp[now][i]=token[i];
```

```
    for(int len=2;len<=N;len++) {
        last=now;
        now=1-now;
        int lim=N-len+1;
        Q_in.clear();

        for(int i=1; i<=len && i<=lim+1 ;i++) {
            while(!Q_in.empty() && Q_in.back().st<dp[last][i]) Q_in.p
op_back();
            Q_in.emplace_back(dp[last][i],i+len-2);
        }

        for(int i=1;i<=lim+2;i++) dp[now][i]=0;

        for(int l=1;l<=lim;l++) {
            int r=l+len-1;

            while(!Q_in.empty() && Q_in.front().nd<l) Q_in.pop_front(
);
            if(r<=lim+1) {
                while(!Q_in.empty() && Q_in.back().st<dp[last][r]) Q_
in.pop_back();
                Q_in.emplace_back(dp[last][r],r+len-2);
            }
            assert(!Q_in.empty());

            dp[now][l]=Q_in.front().st+(qs[r]-qs[l-1])/2; // max_in

        }
    }

    cout << dp[now][1];
    return ;
}
```

Time Complexity: $\mathcal{O}(N^2)$

Subtask #5 : $C_i = i$

จะสังเกตเห็นว่าลำดับของถุงเงินเป็นแบบ Strictly Increasing Sequence ซึ่งเราจะ Observe ได้ว่าการเลือกโกยถุงเงินที่อยู่แถบทางขวาย่อมได้เงินมากกว่าทางซ้าย แต่อย่างไรก็ตามเราก็ต้องพยายามไม่ให้เงินที่ได้รับโดนหักด้วยเช่นกัน ฉะนั้นการเลือกซ้ายขวาสลับไปมาก็ยังถือเป็นวิธีที่ดีที่สุดเช่นเดิม เราสามารถใช้ Decrease and Conquer แก้ปัญหานี้ได้

```
const int Size=5007;
long long token[Size],qs[Size],ans;

void decrease(int N,int dif) {
    if((N-dif)%2==0) {
        decrease(N,dif+1);
        return ;
    }
    if(dif>=N) return ;
    ans+=qs[N]-qs[dif];
    decrease(N,dif+4);
    return ;
}

void process(void) {
    int N;
    cin >> N;
    for(int i=1;i<=N;i++) cin >> token[i];
    for(int i=1;i<=N;i++) qs[i]=qs[i-1]+token[i];

    int observe = (N+3)/2;

    for(int i=observe;i<=N;i++) ans+=(qs[N]-qs[N-i])/2;
    decrease(N,0);

    cout << ans;
    return ;
}
```

Time Complexity: $\mathcal{O}(N)$

Subtask #6 : ไม่มีเงื่อนไขเพิ่มเติม

วิธีทำ Subtask สุดท้ายนี้คือเราต้องนำ dp ที่ใช้เก็บ max_{left} กับ max_{right} มาช่วยลดเวลาในการทำงานของ max_{out} ส่วนตัวของ max_{in} สามารถนำวิธีคิดของ Subtask #4 มาใช้ได้เลย เท่านี้ก็จะได้ระยะเวลาการทำงานที่รวดเร็วพอสำหรับข้อนี้แล้ว

Algorithm: Sliding Window + Dynamic Programming

```
#include <bits/stdc++.h>
#define DEBUG 0
#define st first
#define nd second
#define ll long long
```

```
#define mp make_pair
#define pi pair
using namespace std;
typedef double db;
const int SZ=2e5+3;
const int MX=1e9+7;
const int MOD=1e9+7;

const int Size=5007;
ll token[Size],qs[Size],max_left[Size],max_right[Size],dp[2][Size];

void process(void) {
    int N;
    cin >> N;
    assert(N<=5000);

    for(int i=1;i<=N;i++) cin >> token[i];
    for(int i=1;i<=N;i++) qs[i]=qs[i-1]+token[i];

    int now=1,last=0;
    deque <pair<ll,int>> Q_in;

    for(int i=1;i<=N;i++) dp[now][i]=token[i]; // len=1

    for(int len=2;len<=N;len++) {
        last=now;
        now=1-now;
        int lim=N-len+1;
        Q_in.clear();

        for(int i=1; i<=len && i<=lim+1 ;i++) {
            while(!Q_in.empty() && Q_in.back().st<dp[last][i]) Q_in.p
op_back();
            Q_in.emplace_back(dp[last][i],i+len-2);
        }

        for(int i=1;i<=lim+2;i++) { // Initial
            dp[now][i]=0;
            max_left[i]=0;
            max_right[i]=0;
        }

        for(int i=1;i<=lim+1;i++) max_left[i]=max(max_left[i-1],dp[la
st][i]);
```

```
        for(int i=lim+1;i>=1;i--) max_right[i]=max(max_right[i+1],dp[
last][i]);

        for(int l=1;l<=lim;l++) {
            int r=l+len-1;
            ll max_out=-1;

            while(!Q_in.empty() && Q_in.front().nd<l) Q_in.pop_front(
);
            if(r<=lim+1) {
                while(!Q_in.empty() && Q_in.back().st<dp[last][r]) Q_
in.pop_back();
                Q_in.emplace_back(dp[last][r],r+len-2);
            }
            assert(!Q_in.empty());

            if(l-len+1>0) max_out=max(max_out,max_left[l-len+1]);
            if(r+len-1<=N) max_out=max(max_out,max_right[r+1]);

            dp[now][l]=Q_in.front().st+(qs[r]-qs[l-1])/2; // max_in
            if(max_out!=-1) dp[now][l]=max(dp[now][l],max_out+(qs[r]-
qs[l-1])); // max_out

        }
    }

    cout << dp[now][1];
    return ;
}

int main() {
    #ifndef DEBUG
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    #endif
    cin.tie(nullptr)->ios::sync_with_stdio(false);
    int t(1);
    while(t--) process();
}
```

Time Complexity: $\mathcal{O}(N^2)$