Week 18: Advanced Dynamic Programming & Optimization Techniques

Topics: - Divide-and-Conquer DP - Convex Hull Trick (CHT) for DP optimization - Monotonic Queue / Deque optimization - Knuth Optimization - DP with Bitmask / State Compression

Weekly Tips: - Identify DP recurrences that can be optimized using divide-and-conquer. - Convex Hull Trick reduces O(n^2) DP to O(n log n) for linear functions. - Monotonic queues help optimize sliding window or consecutive state DP. - Knuth Optimization applies when the DP recurrence has quadrangle inequality. - Bitmask DP is effective for small n (<=20) subsets problems.

Problem 1: Convex Hull Trick DP Link: Codeforces 321C **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Line {
    long long m, b;
    long long eval(long long x){ return m*x+b; }
};
vector<Line> hull;
bool bad(Line l1, Line l2, Line l3){
    return (l3.b - l1.b)*(l1.m - l2.m) <= (l2.b - l1.b)*(l1.m - l3.m);
}
void add(Line l){
    while(hull.size()>=2 && bad(hull[hull.size()-2], hull[hull.size()-1], l))
hull.pop_back();
    hull.push_back(l);
}
long long query(long long x){
    int l=0, r=hull.size()-1;
    while(l<r){
        int m=(l+r)/2;
        if(hull[m].eval(x)<=hull[m+1].eval(x)) l=m+1; else r=m;
    }
    return hull[l].eval(x);
}
int main(){
    int n; cin>>n;
    vector<long long> a(n+1), dp(n+1);
    for(int i=1;i<=n;i++) cin>>a[i];
    dp[1]=0;
    add({-2*a[1], a[1]*a[1]+dp[1]});
    for(int i=2;i<=n;i++){
        dp[i] = a[i]*a[i] + query(a[i]);
        add({-2*a[i], a[i]*a[i]+dp[i]});
```

```
    }
    cout<<dp[n]<<endl;
}
```

Explanation Comments: - Convex Hull Trick optimizes DP of form `dp[i] = min(dp[j] + a[i]*a[i] + f(j))`. - Lines represent previous DP states; binary search finds minimum efficiently. - Reduces naive O(n^2) DP to O(n log n).

Problem 2: Monotonic Queue Optimization Link: CSES Sliding Cost **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
deque<int> dq;
int main(){
    int n,k; cin>>n>>k;
    vector<int> a(n);
    for(int i=0;i<n;i++) cin>>a[i];
    vector<int> dp(n);
    for(int i=0;i<n;i++){
        while(!dq.empty() && dq.front() < i-k) dq.pop_front();
        dp[i] = a[i] + (dq.empty()?0:dp[dq.front()]);
        while(!dq.empty() && dp[i] <= dp[dq.back()]) dq.pop_back();
        dq.push_back(i);
    }
    cout<<dp[n-1]<<endl;
}
```

Explanation Comments: - Monotonic queue maintains candidates for DP in sliding window. - Ensures O(n) update per state instead of O(k). - Useful for interval/segment DP problems with constraints.

End of Week 18 - Master DP optimization techniques to handle large states efficiently. - Understand when and how to apply CHT, Monotonic Queue, Divide-and-Conquer, and Knuth Optimization. - Practice with multiple variants to build intuition for ACM-ICPC challenges.