CS35X: Competitive Programming

Lecture 9: Prefix Sums, Binary Search The Answer

Joshua Brody

Warmup Kattis Problem: Kuggfragan

Problem debrief: bigtruck

Sample Problem: Range Sums

- Input: array **A** of up to 100000 integers
  - **A = [-2 15 -14 60 -5 1]**
- Goal: compute maximal value of a subarray A[I…j]: **A[i] + … + A[j]**

Sample Problem: Range Sums

- Input: array **A** of up to 100000 integers
  - **A = [-2 15 -14 60 -5 1]**
- Goal: compute maximal value of a subarray A[i…j]: **A[i] + … + A[j]**
- **A[1..3] = 15-14+60 = 61**

Sample Problem: Range Sums

- Input: array **A** of up to 100000 integers
  - **A = [-2 15 -14 60 -5 1]**
- Goal: compute maximal value of a subarray A[i…j]: **A[i] + … + A[j]**
- **A[1..3] = 15-14+60 = 61**
- Brute Force:
  - for each (i,j), compute **A[i]+…+A[j],** return maximal sum
  - **O(n^3)** time

Prefix Sums: range sums in O(n) time

- Given an array A, the prefix sum of A is defined such that
  - pre[i] is the sum of the first i elements of A.

```
pre[0]=0;
for i = 1…N:
  pre[i] = pre[i-1]+A[i-1];
```

Prefix Sums: range sums in O(n) time

- Given an array A, the prefix sum of A is defined such that
  - pre[i] is the sum of the first i elements of A.

```
pre[0]=0;
for i = 1…N:
  pre[i] = pre[i-1]+A[i-1];
```

- We can use prefix sums to compute range sums:

```
lo_pref_sum=0;
hi_rangesum=0;
for i =1..N:
  hi_rangesum = max(hi_rangesum, pre[i]-lo_pref_sum)
  lo_pref_sum = min(lo_pref_sum, pre[i])
```

Kattis Problem: Commercials

Recap: Binary Search

- Input: sorted array **A** of numbers and number **x** to search for
  - **A = [1 2 5 8 19 21 28 34 39 42]**
  - **x = 34**
- Goal: return index of **x** in **A**, or **-1** if **x** not in **A**
  - **search(A,x) = 7**

Recap: Binary Search

- Input: sorted array **A** of numbers and number **x** to search for
  - **A = [1 2 5 8 19 21 28 34 39 42]**
  - **x = 34**
- Goal: return index of **x** in **A**, or **-1** if **x** not in **A**
  - **search(A,x) = 7**
- Binary search:
  - query middle index, return index if **A[mid]==x**
  - if **x< A[mid]**, search left half
  - if **x > A[mid]**, search right half

Binary Search The Answer (BSTA)

- Maybe optimization problem is hard to solve, but ***easy*** if you know the answer to look for.
- If you can easily check if an answer is feasible, then you can use binary search on the answer to check the best answer
-

```
// problem-specific range of possible values for answer
lo=0, high=1000000;
while(lo < high) {
  mid=(lo+high)/2;
  if(AnswerAtMost(mid)) {
    high = mid;
  else {
    lo = mid+1;
  }
}
```

Kattis Problem: freeweights