Algorithm design for ACM-ICPC

Taejin Chin, Wonha Ryu

KAIST

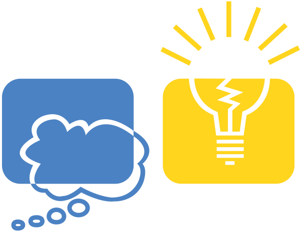
Winter School on Algorithms and Combinatorics 2010

ACM-ICPC

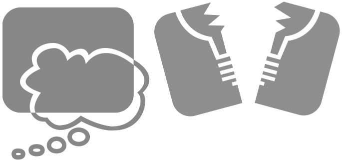
acm International Collegiate Programming Contest

ACM-ICPC

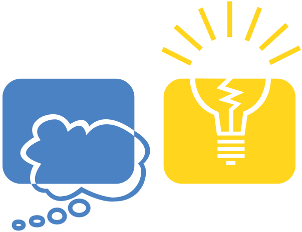
ACM-ICPC



ACM-ICPC



ACM-ICPC



ACM-ICPC

ACM-ICPC

- World's largest programming contest - over 7000 teams!
- 3 students compete in a team, using only 1 machine.
- About 10 problems and 5 hours are given.
- Judgement is in real-time. A balloon is awarded for accepted solutions.

ACM-ICPC

- World's largest programming contest - over 7000 teams!
- 3 students compete in a team, using only 1 machine.
- About 10 problems and 5 hours are given.
- Judgement is in real-time. A balloon is awarded for accepted solutions.

ACM-ICPC

- World's largest programming contest - over 7000 teams!
- 3 students compete in a team, using only 1 machine.
- About 10 problems and 5 hours are given.
- Judgement is in real-time. A balloon is awarded for accepted solutions.

ACM-ICPC

- World's largest programming contest - over 7000 teams!
- 3 students compete in a team, using only 1 machine.
- About 10 problems and 5 hours are given.
- Judgement is in real-time. A balloon is awarded for accepted solutions.

ACM-ICPC

- World's largest programming contest - over 7000 teams!
- 3 students compete in a team, using only 1 machine.
- About 10 problems and 5 hours are given.
- Judgement is in real-time. A balloon is awarded for accepted solutions.

ACM-ICPC

- World's largest programming contest - over 7000 teams!
- 3 students compete in a team, using only 1 machine.
- About 10 problems and 5 hours are given.
- Judgement is in real-time. A balloon is awarded for accepted solutions.

Picture from World Finals 2008-2009

Problem: Room Assignments

WSAC 2011 is going to be held, from last year's success.

Lastly, Wonha has been asked to assign rooms to participants.

But in this time, we're able to assign exactly one room for each participant.

Organizer has reserved rooms as many as the number of the participants.

Problem: Room Assignments

WSAC 2011 is going to be held, from last year's success.

Lastly, Wonha has been asked to assign rooms to participants.

But in this time, we're able to assign exactly one room for each participant.

Organizer has reserved rooms as many as the number of the participants.

Problem: Room Assignments (cont.)

So he designed sophiscated but fair way to assign rooms.

At the registration time, every participant chooses two preferred rooms, and writes down their numbers on both side of the coin.

And then, everyone throws their own coin.

If there is no conflict, assignment is done. . . but otherwise, everyone should re-throw their coin until assignment is done.

This approach ensures uniformness of the assignment!

Problem: Room Assignments (cont.)

So he designed sophiscated but fair way to assign rooms.

At the registration time, every participant chooses two preferred rooms, and writes down their numbers on both side of the coin.

And then, everyone throws their own coin.

If there is no conflict, assignment is done... but otherwise, everyone should re-throw their coin until assignment is done.

This approach ensures uniformness of the assignment!

Problem: Room Assignments (cont.)

So he designed sophiscated but fair way to assign rooms.

At the registration time, every participant chooses two preferred rooms, and writes down their numbers on both side of the coin.

And then, everyone throws their own coin.

If there is no conflict, assignment is done. . . but otherwise, everyone should re-throw their coin until assignment is done.

This approach ensures uniformness of the assignment!

Problem: Room Assignments (cont.)

Wonha, a little undergraduate, is also participant too.

He already rated the preference of each room in a numerical way.

Formally speaking, he rated the preference of room k by integer v_k.

As a moderator, he knows all others' preference; so he's going to have some advantage!

Help him to maximize expected rating of assigned room.

Problem: Room Assignments (cont.)

Wonha, a little undergraduate, is also participant too.

He already rated the preference of each room in a numerical way.

Formally speaking, he rated the preference of room k by integer v_k.

As a moderator, he knows all others' preference; so he's going to have some advantage!

Help him to maximize expected rating of assigned room.

Problem: Room Assignments (cont.)

Input

```
C               # Number of test cases (1 <= C <= 200)
N               # Number of rooms (2 <= N <= 50000)
a_1 b_1         # Preferred two rooms
a_2 b_2         # 1 <= a_k < b_k <= N
..
a_N-1 b_N-1
v_1 v_2 .. v_N # Rating (1 <= v_k <= 1000000)
```

Problem: Room Assignments (cont.)

Output

For each test case,

- Print a single line containing the two different room numbers a and b, which should be selected by Wonha.
- If there is more than one optimal selection, break ties by choosing lexicographically smallest solution.
- If there is no way to select two rooms such that an assignment is possible, print "impossible" instead.

Modeling

We're going to model the objects using graph.

- Vertex - each room.
- (undirected) Edge - each person.
  - Endpoint of edge - selected rooms.
  - Direction of edge - result of coin toss.

Modeling

We're going to model the objects using graph.

- Vertex - each room.
- (undirected) Edge - each person.
  - Endpoint of edge - selected rooms.
  - Direction of edge - result of coin toss.

Modeling

We're going to model the objects using graph.

- Vertex - each room.
- (undirected) Edge - each person.
  - Endpoint of edge - selected rooms.
  - Direction of edge - result of coin toss.

Modeling

We're going to model the objects using graph.

- Vertex - each room.
- (undirected) Edge - each person.
  - Endpoint of edge - selected rooms.
  - Direction of edge - result of coin toss.

Modeling

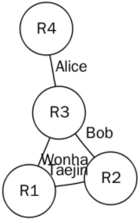
We're going to model the objects using graph.

- Vertex - each room.
- (undirected) Edge - each person.
  - Endpoint of edge - selected rooms.
  - Direction of edge - result of coin toss.

Modeling

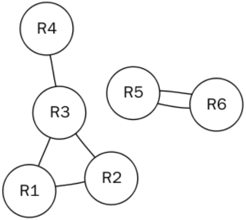
We're going to model the objects using graph.

- Vertex - each room.
- (undirected) Edge - each person.
  - Endpoint of edge - selected rooms.
  - Direction of edge - result of coin toss.



Observation: independence

Two disconnected components are independent.



We should focus on each component.

Observation: classifying components

- $V = E + 1$ ... tree!
- $V = E$ ...?
- $V < E$ ... by pigeonhole principle, assignment is impossible.

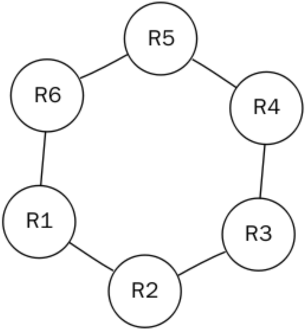
Observation: classifying components

- $V = E + 1$ ...tree!
- $V = E$ ...?
- $V < E$ ...by pigeonhole principle, assignment is impossible.

Observation: classifying components

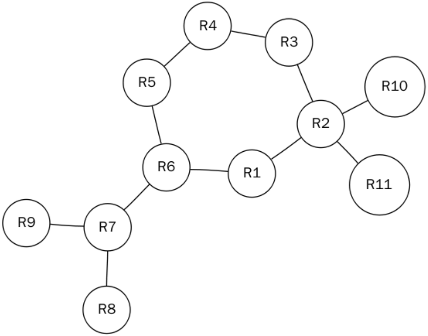
- $V = E + 1$ ... tree!
- $V = E$ ... ?
- $V < E$ ... by pigeonhole principle, assignment is impossible.

Observation: case of $V = E$



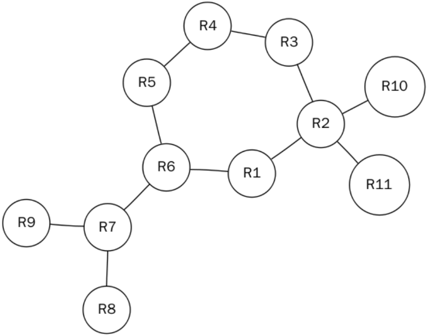
Only a single cycle exists!

Observation: case of $V = E$



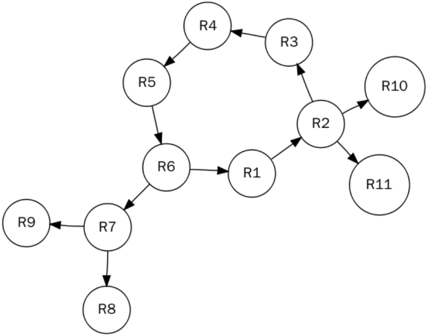
Only a single cycle exists!

Observation: case of $V = E$



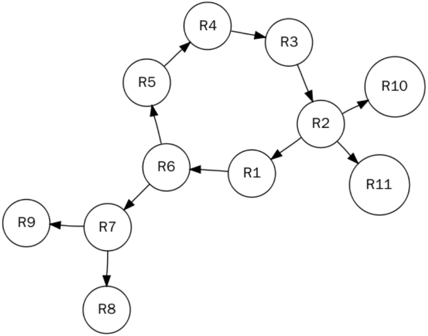
Only a single cycle exists!

Observation: case of $V = E$



Only a single cycle exists!

Observation: case of $V = E$



Only a single cycle exists!

Reinterpreting the problem

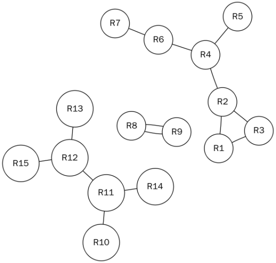
Without my selection, $V = N, E = N - 1$. For example:

We're going to add a single edge.

(Note: there exists one and only tree component for every "valid" graph.)

Reinterpreting the problem

Without my selection, $V = N, E = N - 1$. For example:

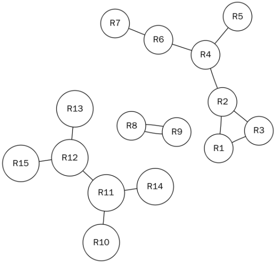


We're going to add a single edge.

(Note: there exists one and only tree component for every "valid" graph.)

Reinterpreting the problem

Without my selection, $V = N, E = N - 1$. For example:

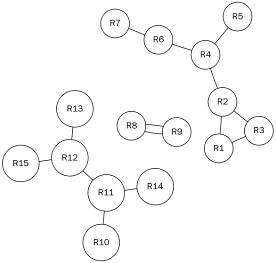


We're going to add a single edge.

(Note: there exists one and only tree component for every "valid" graph.)

Reinterpreting the problem

Without my selection, $V = N, E = N - 1$. For example:



We're going to add a single edge.

(Note: there exists one and only tree component for every "valid" graph.)

Separating the cases. . . done!

- Tree-to-Tree: A cycle is generated. Each of our endpoint rooms will have equal probability.
- Tree-to-Other: One endpoint will never be selected. Expectation is equal to other endpoint's rating.
- Other-to-Other: impossible!

Separating the cases. . . done!

- Tree-to-Tree: A cycle is generated. Each of our endpoint rooms will have equal probability.
- Tree-to-Other: One endpoint will never be selected. Expectation is equal to other endpoint's rating.
- Other-to-Other: impossible!

Separating the cases. . . done!

- Tree-to-Tree: A cycle is generated. Each of our endpoint rooms will have equal probability.
- Tree-to-Other: One endpoint will never be selected. Expectation is equal to other endpoint's rating.
- Other-to-Other: impossible!

Separating the cases. . . done!

- Tree-to-Tree: A cycle is generated. Each of our endpoint rooms will have equal probability.
- Tree-to-Other: One endpoint will never be selected. Expectation is equal to other endpoint's rating.
- Other-to-Other: impossible!

Separating the cases. . . done!

- Tree-to-Tree: A cycle is generated. Each of our endpoint rooms will have equal probability.
- Tree-to-Other: One endpoint will never be selected. Expectation is equal to other endpoint's rating.
- Other-to-Other: impossible!

Separating the cases. . . done!

- Tree-to-Tree: A cycle is generated. Each of our endpoint rooms will have equal probability.
- Tree-to-Other: One endpoint will never be selected. Expectation is equal to other endpoint's rating.
- Other-to-Other: impossible!

Problem: "Densest" subsequence

Problem description

From given sequence S with length N,

- Find densest continuous subsequence and its mean
- ...with length at least L.

Variations

- Submatrix version (Taiwan 2001)
- Binary sequence version (Seoul 2009)

Problem: "Densest" subsequence

Problem description

From given sequence S with length N,

- Find densest continuous subsequence and its mean
- ...with length at least L.

Variations

- Submatrix version (Taiwan 2001)
- Binary sequence version (Seoul 2009)

Problem: "Densest" subsequence

Problem description

From given sequence S with length N,

- Find densest continuous subsequence and its mean
- . . . with length at least L.

Variations

- Submatrix version (Taiwan 2001)
- Binary sequence version (Seoul 2009)

Problem: "Densest" subsequence

Problem description

From given sequence S with length N,

- Find densest continuous subsequence and its mean
- ...with length at least L.

Variations

- Submatrix version (Taiwan 2001)
- Binary sequence version (Seoul 2009)

Problem: "Densest" subsequence

Problem description

From given sequence S with length N,

- Find densest continuous subsequence and its mean
- ...with length at least L.

Variations

- Submatrix version (Taiwan 2001)
- Binary sequence version (Seoul 2009)

Problem: "Densest" subsequence

Problem description

From given sequence S with length N,

- Find densest continuous subsequence and its mean
- ... with length at least L.

Variations

- Submatrix version (Taiwan 2001)
- Binary sequence version (Seoul 2009)

Naïve approaches

Most simple approach

Try every possible subsequence . . . leads to $O(n^3)$. Infeasible.

Yet inefficient approach

- Let's define $\mu_{i,j}$ as mean of subsequence $S[i \dots j]$ (inclusive).
- Evaluating $\mu_{i,j+1}$ or $\mu_{i+1,j+1}$ from $\mu_{i,j}$ isn't difficult.
- It leads overall time complexity to $O(n^2)$.

Naïve approaches

Most simple approach

Try every possible subsequence . . . leads to $O(n^3)$. Infeasible.

Yet inefficient approach

- Let's define $\mu_{i,j}$ as mean of subsequence $S[i \dots j]$ (inclusive).
- Evaluating $\mu_{i,j+1}$ or $\mu_{i+1,j+1}$ from $\mu_{i,j}$ isn't difficult.
- It leads overall time complexity to $O(n^2)$.

Naïve approaches

Most simple approach

Try every possible subsequence . . . leads to $O(n^3)$. Infeasible.

Yet inefficient approach

- Let's define $\mu_{i,j}$ as mean of subsequence $S[i \dots j]$ (inclusive).
- Evaluating $\mu_{i,j+1}$ or $\mu_{i+1,j+1}$ from $\mu_{i,j}$ isn't difficult.
- It leads overall time complexity to $O(n^2)$.

Naïve approaches

Most simple approach

Try every possible subsequence . . . leads to $O(n^3)$. Infeasible.

Yet inefficient approach

- Let's define $\mu_{i,j}$ as mean of subsequence $S[i \dots j]$ (inclusive).
- Evaluating $\mu_{i,j+1}$ or $\mu_{i+1,j+1}$ from $\mu_{i,j}$ isn't difficult.
- It leads overall time complexity to $O(n^2)$.

Naïve approaches

Most simple approach

Try every possible subsequence ... leads to $O(n^3)$. Infeasible.

Yet inefficient approach

- Let's define $\mu_{i,j}$ as mean of subsequence $S[i \dots j]$ (inclusive).
- Evaluating $\mu_{i,j+1}$ or $\mu_{i+1,j+1}$ from $\mu_{i,j}$ isn't difficult.
- It leads overall time complexity to $O(n^2)$.

Naïve approaches

Most simple approach

Try every possible subsequence . . . leads to $O(n^3)$. Infeasible.

Yet inefficient approach

- Let's define $\mu_{i,j}$ as mean of subsequence $S[i \dots j]$ (inclusive).
- Evaluating $\mu_{i,j+1}$ or $\mu_{i+1,j+1}$ from $\mu_{i,j}$ isn't difficult.
- It leads overall time complexity to $O(n^2)$.

Approach for small L

- Key observation:
  - For a subsequence $S[i \ldots j]$, with length $\geq 2L$.
  - Split that one into two subsequences, like $\mu_{i,i+L-1}$ and $\mu_{i+L,j}$
  - One of them has bigger mean.
- ...leads to $O(NL)$, infeasible for $L = O(N)$.
- But for Seoul regional problem, $L \ll N$: many team(almost all?) solved it in this way.

Approach for small L

- Key observation:
  - For a subsequence $S[i \dots j]$, with length $\geq 2L$.
  - Split that one into two subsequences, like $\mu_{i,i+L-1}$ and $\mu_{i+L,j}$
  - One of them has bigger mean.
- …leads to $O(NL)$, infeasible for $L = O(N)$.
- But for Seoul regional problem, $L \ll N$: many team(almost all?) solved it in this way.

Approach for small L

- Key observation:
  - For a subsequence $S[i \dots j]$, with length $\geq 2L$.
  - Split that one into two subsequences, like $\mu_{i,i+L-1}$ and $\mu_{i+L,j}$
  - One of them has bigger mean.
- …leads to $O(NL)$, infeasible for $L = O(N)$.
- But for Seoul regional problem, $L \ll N$: many team(almost all?) solved it in this way.

Approach for small L

- Key observation:
  - For a subsequence $S[i \dots j]$, with length $\geq 2L$.
  - Split that one into two subsequences, like $\mu_{i,i+L-1}$ and $\mu_{i+L,j}$
  - One of them has bigger mean.
- ...leads to $O(NL)$, infeasible for $L = O(N)$.
- But for Seoul regional problem, $L \ll N$: many team(almost all?) solved it in this way.

Approach for small L

- Key observation:
  - For a subsequence $S[i \dots j]$, with length $\geq 2L$.
  - Split that one into two subsequences, like $\mu_{i,i+L-1}$ and $\mu_{i+L,j}$
  - One of them has bigger mean.
- ...leads to $O(NL)$, infeasible for $L = O(N)$.
- But for Seoul regional problem, $L \ll N$: many team(almost all?) solved it in this way.

Approach for small L

- Key observation:
  - For a subsequence $S[i \dots j]$, with length $\geq 2L$.
  - Split that one into two subsequences, like $\mu_{i,i+L-1}$ and $\mu_{i+L,j}$
  - One of them has bigger mean.
- …leads to $O(NL)$, infeasible for $L = O(N)$.
- But for Seoul regional problem, $L \ll N$: many team(almost all?) solved it in this way.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences? $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'} (m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences?
    $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'} (m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences?
    $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'} (m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences?
    $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'}(m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences? $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'} (m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences? $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'} (m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Reinterpreting the problem

- Maximize $\frac{\sum_{i \in S'} S[i]}{|S'|}$, among all possible subsequence S'.
- Converting the problem into decision problem:
  - Is m upper bound for means of all subsequences? $m \geq \frac{\sum_{i \in S'} S[i]}{\sum_{i \in S'} 1}$
  - In other words, $f(S') = \sum_{i \in S'}(m - S[i]) \geq 0$. We have to find minimum of $f(S')$.
  - If we define $T[i] := m - S[i]$, the problem is essentially same as minimum sum subsequence problem. Easy.
- We can binary search over all possible means, until sufficient precision is acquired.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: "base window"
- Maintain "blocks" ordered by its mean like this:
- Include some blocks into subsequence, if included subsequence results bigger mean.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: "base window"
- Maintain "blocks" ordered by its mean like this:
- Include some blocks into subsequence, if included subsequence results bigger mean.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: "base window"
- Maintain "blocks" ordered by its mean like this:
- Include some blocks into subsequence, if included subsequence results bigger mean.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: "base window"
- Maintain "blocks" ordered by its mean like this:

- Include some blocks into subsequence, if included subsequence results bigger mean.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: ”base window”
- Maintain ”blocks” ordered by its mean like this:

- Include some blocks into subsequence, if included subsequence results bigger mean.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: "base window"
- Maintain "blocks" ordered by its mean like this:

- Include some blocks into subsequence, if included subsequence results bigger mean.

Basic idea

- Iterate for an endpoint of subsequence.
- L elements from the endpoint must be included: "base window"
- Maintain "blocks" ordered by its mean like this:

- Include some blocks into subsequence, if included subsequence results bigger mean.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.
- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.
- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.

- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.

- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.

- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.

- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks

Removing blocks from the list

- Remove first block when it's better not to use.

- Why? These blocks are never useful, because we're looking for maximum!
- Each block is removed at most once.

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Managing the list of blocks (cont.)

Inserting new element into the list

- When base window advances, one element will be added into list.

- Merge some rear blocks to be in order of its means like this:
- Shifting base window increments the number of blocks.
- Merging two blocks decrements the number of blocks.
- Using data structures like deque, we achieve $O(N)$!

Google Code Jam

- Renewed from 2008.
- Anyone with age 13+ is eligible to compete.
- Annual event track consists of several rounds: some online rounds, local office onsites, onsite finals at Mountain View.
- 3 to 6 problems are given in 2 hours to 4 hours.
- Any language can be used. Competitor downloads input, runs it in one's own machine, and uploads the result with code.

Google Code Jam

- Renewed from 2008.
- Anyone with age 13+ is eligible to compete.
- Annual event track consists of several rounds: some online rounds, local office onsites, onsite finals at Mountain View.
- 3 to 6 problems are given in 2 hours to 4 hours.
- Any language can be used. Competitor downloads input, runs it in one's own machine, and uploads the result with code.

Google Code Jam

- Renewed from 2008.
- Anyone with age 13+ is eligible to compete.
- Annual event track consists of several rounds: some online rounds, local office onsites, onsite finals at Mountain View.
- 3 to 6 problems are given in 2 hours to 4 hours.
- Any language can be used. Competitor downloads input, runs it in one's own machine, and uploads the result with code.

Google Code Jam

- Renewed from 2008.
- Anyone with age 13+ is eligible to compete.
- Annual event track consists of several rounds: some online rounds, local office onsites, onsite finals at Mountain View.
- 3 to 6 problems are given in 2 hours to 4 hours.
- Any language can be used. Competitor downloads input, runs it in one's own machine, and uploads the result with code.

Google Code Jam

- Renewed from 2008.
- Anyone with age 13+ is eligible to compete.
- Annual event track consists of several rounds: some online rounds, local office onsites, onsite finals at Mountain View.
- 3 to 6 problems are given in 2 hours to 4 hours.
- Any language can be used. Competitor downloads input, runs it in one's own machine, and uploads the result with code.

Photo from Google Code Jam 2008

Introducing TopCoder

- TopCoder provides various competition platform, including "Algorithm Match" and "Marathon Match".
- Each member's performance is rated in a numerical way, like ladder system of Battle.net.
- Hosts annual competition track named TopCoder Open.

Introducing TopCoder

- TopCoder provides various competition platform, including "Algorithm Match" and "Marathon Match".
- Each member's performance is rated in a numerical way, like ladder system of Battle.net.
- Hosts annual competition track named TopCoder Open.

Introducing TopCoder

- TopCoder provides various competition platform, including "Algorithm Match" and "Marathon Match".
- Each member's performance is rated in a numerical way, like ladder system of Battle.net.
- Hosts annual competition track named TopCoder Open.

Photo from TopCoder Open 2009

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Algorithm Match

- The match consists of several phases:
  - 75 minutes for coding phase,
  - 5 minutes for intermission,
  - 15 minutes for challenge phase.
- Three problems with score weighted on their difficulty are given.
- Faster submission, higher score.
- Submissions are judged against preconstructed test cases, like ACM-ICPC.

Marathon Match

- Problems with no optimal solutions are given.
- Scoring mechanism may be different for each problem.
- Some problems have "real-world" purpose: winners may be awarded by prize.
- Recent example:
  - Cracking Enigma, hosted by NSA.
  - Packaging a medkit, hosted by NASA.

Marathon Match

- Problems with no optimal solutions are given.
- Scoring mechanism may be different for each problem.
- Some problems have "real-world" purpose: winners may be awarded by prize.
- Recent example:
  - Cracking Enigma, hosted by NSA.
  - Packaging a medkit, hosted by NASA.

Marathon Match

- Problems with no optimal solutions are given.
- Scoring mechanism may be different for each problem.
- Some problems have "real-world" purpose: winners may be awarded by prize.
- Recent example:
  - Cracking Enigma, hosted by NSA.
  - Packaging a medkit, hosted by NASA.

Marathon Match

- Problems with no optimal solutions are given.
- Scoring mechanism may be different for each problem.
- Some problems have "real-world" purpose: winners may be awarded by prize.
- Recent example:
  - Cracking Enigma, hosted by NSA.
  - Packaging a medkit, hosted by NASA.

Marathon Match

- Problems with no optimal solutions are given.
- Scoring mechanism may be different for each problem.
- Some problems have "real-world" purpose: winners may be awarded by prize.
- Recent example:
  - Cracking Enigma, hosted by NSA.
  - Packaging a medkit, hosted by NASA.

Marathon Match

- Problems with no optimal solutions are given.
- Scoring mechanism may be different for each problem.
- Some problems have "real-world" purpose: winners may be awarded by prize.
- Recent example:
  - Cracking Enigma, hosted by NSA.
  - Packaging a medkit, hosted by NASA.