Advanced Greedy

Ethen Yuen {ethening}
2023-07-02

Table of Contents

Background

- What is Greedy?

Background

- What is Greedy?
- We take the best step in the **current situation** for every step
  - Opt for “Instant Gratification”

Background

- What is Greedy?
- We take the best step in the **current situation** for every step
  - Opt for “Instant Gratification”

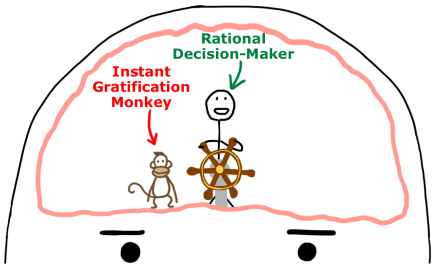


Figure: Instant Gratification Monkey Illustration by Tim Urban

Background

- What is Greedy?
- We take the best step in the **current situation** for every step
  - Opt for “Instant Gratification”

Figure: Instant Gratification Monkey Illustration by Tim Urban

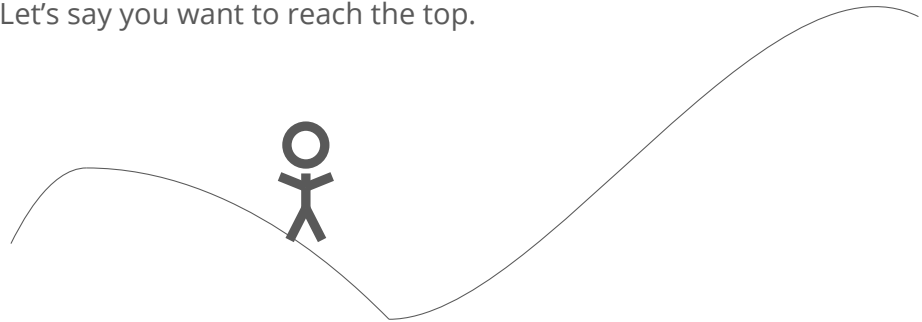
Background

- What is Greedy?
- We take the best step in the **current situation** for every step
  - Opt for “Instant Gratification”
  - *May* lead you to somewhere good
  - Not guaranteed to get you to the optimal answer

Figure: Instant Gratification Monkey Illustration by Tim Urban

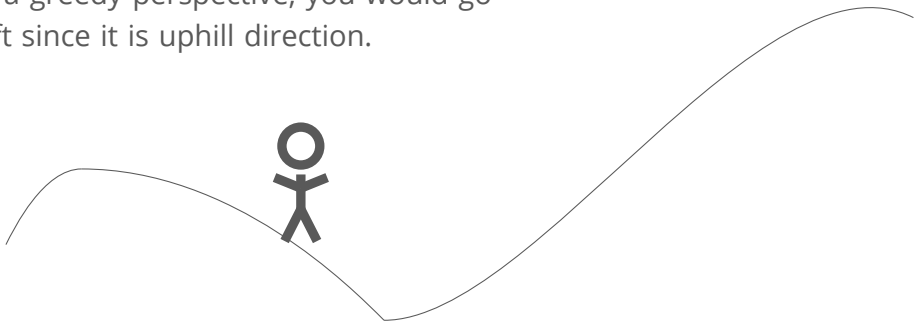
What's wrong with Greedy?

Let's say you want to reach the top.



What's wrong with Greedy?

In a greedy perspective, you would go left since it is uphill direction.

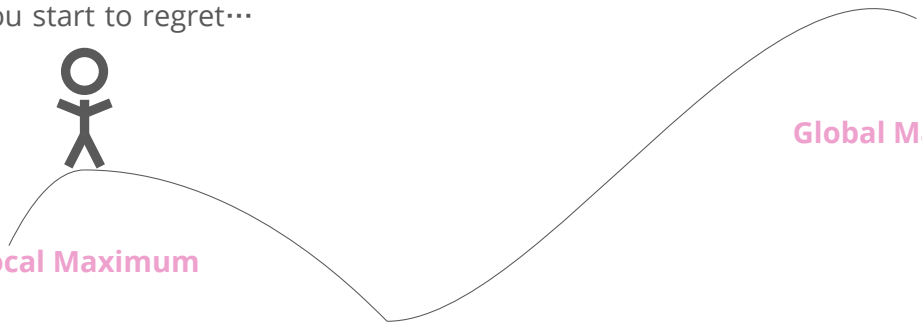


What's wrong with Greedy?

Sadly, now both sides are downhill, so you are stuck in some non-optimal place.

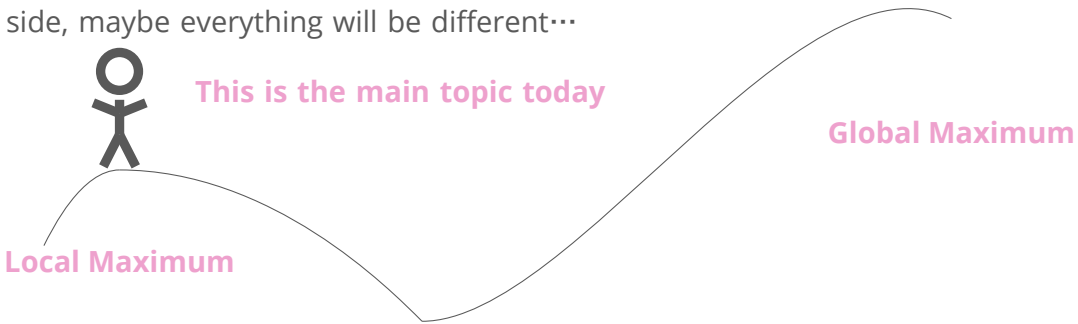
What's wrong with Greedy?

You start to regret…



What's wrong with Greedy?

If you can undo your move and go to the other side, maybe everything will be different…



Fixing Greedy

- Normally, Greedy only works when taking the best step in any situation will never lead to a sub-optimal solution.
- When taking the best step will go wrong, given that we can *undo* the wrong step after realizing it, we can still go back to the optimal solution.
- When undoing is possible, we can perform regrettable greedy（反悔貪心）with a regret strategy（反悔策略）.

Table of Contents

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$.

- Answer is obviously $9 + 5 + 4 = 18$.
- This can be easily solved with any fast-enough sorting algorithm.
- But let's approach this problem with Greedy.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3\}$

Let's iterate through the elements one by one.

- When $i = 1$, we pick $A[1] = 3$ because it increases our current sum.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3, 1\}$

Let's iterate through the elements one by one.

- When $i = 1$, we pick $A[1] = 3$ because it increases our current sum.
- When $i = 2$, we pick $A[2] = 1$ because it increases our current sum.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3, 1, 4\}$

Let's iterate through the elements one by one.

- When $i = 1$, we pick $A[1] = 3$ because it increases our current sum.
- When $i = 2$, we pick $A[2] = 1$ because it increases our current sum.
- When $i = 3$, we pick $A[3] = 4$ because it increases our current sum.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3, 1, 4\}$

- When $i = 4$, we need to decide if $A[4] = 5$ should be chosen.
  - We already picked 3 integers.
  - We need to regret choosing an integer that has already been chosen if we want to choose 5.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3, 1, 4\}$

- When $i = 4$, we need to decide if $A[4] = 5$ should be chosen.
  - We already picked 3 integers.
  - We need to regret choosing an integer that has already been chosen if we want to choose 5.
  - Obviously we want to regret picking the smallest integer picked, which is 1.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3, 4, 5\}$

- When $i = 4$, we need to decide if $A[4] = 5$ should be chosen.
  - As $1 < 5$, picking 5 instead is clearly an upgrade.
  - We just drop 1 and get 5 then.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{3, 4, 5\}$

- When $i = 5$, we need to decide if $A[5] = 9$ should be chosen.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{4, 5, 9\}$

- When $i = 5$, we need to decide if $A[5] = 9$ should be chosen.
  - This time, 3 is dropped to make space for 9.

Problem 1

Problem Statement

Given an array $A[1..N]$ filled with positive integers. Find the maximum sum of K distinct elements.

Sample: $A = [3, 1, 4, 5, 9]$, $K = 3$. Currently picked: $\{4, 5, 9\}$

- You can see that, the sum of elements picked is exactly what we expected: $4 + 5 + 9 = 18$.

Problem 1

How do we implement the above method?

- We need to find the `min` of the selected element, to see which step we should regret.
- The finding min part can be optimized by a heap.
- Total time complexity: $O(N \log N)$

Each element in the heap represented **an action to be regretted**. We call this type of algorithm: Heap-regret Greedy (堆反悔貪心) .

Problem 2

Problem Statement

Given N tasks, each task needs a unit time to complete.

Each task has a deadline D_i and a value V_i, solving a task on time t on or before deadline $(t \leq D_i)$ earns you the value of the task.

At each point in time, you can choose to finish a task, what is the maximum value you can get?

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

How to solve this problem?

- We should sort the tasks by ascending order of deadline (then by *descending order of value*), so we would finish the task that is more urgent
- We can then iterate through each task in order.
  If the number of tasks chosen $< D_i$, then choose this task.

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

How to solve this problem?

- We should sort the tasks by ascending order of deadline (then by *descending order of value*), so we would finish the task that is more urgent
- We can then iterate through each task in order.
  If the number of tasks chosen $< D_i$, then choose this task.

Wait, is that really correct?

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

Consider this case:

- $D_1 = 1, V_1 = 1$
- $D_2 = 2, V_2 = 100$
- $D_3 = 2, V_3 = 101$

The greedy algorithm will pick Task 1 and Task 3. However, picking Task 2 and Task 3 is better.

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

- If we fill our schedule doing low-return tasks, we may not be able to finish high-return tasks that are seen later.
- What if, we can *regret* doing some low-return task?

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

Observe that, for any two tasks A and B.

- If $D_A \leq D_B$, and we decide to do A:
  We can use that time to **NOT do A but do B instead** (still valid).

Problem 2

Excerpt: N tasks with deadline D_i and value V_i, each task needs a unit time to complete. Maximize value gained.

Still, iterate in the ascending order of deadline.

- When the number of tasks chosen $= D_i$, it means that the schedule has already been filled up completely.
- We could see if regretting doing the task with the lowest value and doing the current task is better.

It is heap-regret greedy after all.
Time complexity: $O(N \log N)$

Problem 3

Problem Statement

Given N tasks, each task has a deadline D_i and a required time cost T_i to solve.

What is the maximum number of tasks you can solve?

So, this time the value of each task is fixed (which is 1), while the time cost varies.

Problem 3

Excerpt: N tasks with deadline D_i and time cost T_i. Maximize tasks completed.

We can still use the idea of regrettable greedy to solve.

- As we need to do regret action, we want to place ourselves in a position where “regretting” is possible.
- (that means, a position in which you can always choose to do a current action AND undo a previous action)

Problem 3

Excerpt: N tasks with deadline D_i and time cost T_i. Maximize tasks completed.

We can still use the idea of regrettable greedy to solve.

- As we need to do regret action, we want to place ourselves in a position where “regretting” is possible.
- (that means, a position in which you can always choose to do a current action AND undo a previous action)
- We still sort the tasks by ascending order of deadline. (Why?)

Problem 3

Excerpt: N tasks with deadline D_i and time cost T_i. Maximize tasks completed.

We can still use the idea of regrettable greedy to solve.

- We still sort the tasks by ascending order of deadline.
- This time when the schedule is filled up, instead of regretting the task with the lowest value, we regret the task with the **largest time cost**.

Problem 3

Excerpt: N tasks with deadline D_i and time cost T_i. Maximize tasks completed.

We can still use the idea of regrettable greedy to solve.

- We still sort the tasks by ascending order of deadline.
- This time when the schedule is filled up, instead of regretting the task with the lowest value, we regret the task with the **largest time cost**.
- Of course, besides maintaining the heap, you still need to maintain the sum of time cost of the current chosen set of tasks, so it can be used to easily determine if the new task can be inserted without regretting.

Time complexity: $O(N \log N)$

Table of Contents

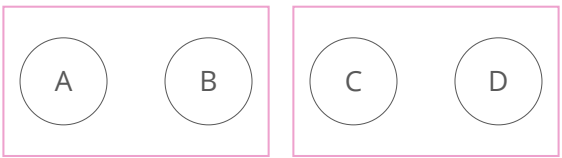
Regretting Automaton

Besides heap-regret greedy, there is something called regretting automaton (反悔自動機) that provides another way for us to do these tasks.

- **Heap-regret greedy:** maintain a heap where each element represents an action to be regretted
- **Regretting automaton**: merge the regret action with the add-new action together by editing the weights of some elements to be chosen. (explain in later slides)
  - Automatically regret when it is better to regret, automatically discard the new element when it is better to keep the current state.
  - Often used on **constrained** decision problem.

Regretting Automaton

Suppose there are 4 integers A, B, C and D. We can only choose 1 of A and B, and choose 1 of C and D. What is the maximum value we can get?

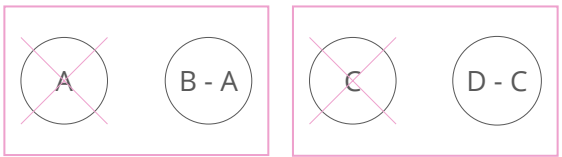


Regretting Automaton

Let's say we have already chosen A and C. (sum $= A + C$).

- We can delete A and C, then turn B into $B - A$, and D into $D - C$.
- Then we can greedily pick each of the remaining nodes ($B - A$, $D - C$) if they are > 0.

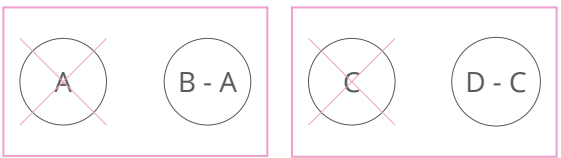
Note that this is turned into an **unconstrained problem**: no restriction on what nodes cannot be chosen together.



Regretting Automaton

Choosing the node $B - A$ here actually is equivalent to choosing B and regretting choosing A. Same with choosing $D - C$.

You can see the regretting action is embedded into the greedy.



Problem 4

Problem Statement

Given the stock price of the following N $(2 \leq N \leq 3 \times 10^5)$ days, $P[1..N]$.
Every day you can:

1. Buy one stock
2. Sell one stock
3. Do nothing

You have 0 stock at the start and need to be with 0 stock at the end. What is the maximum profit?

Link: CF865D - Buy Low Sell High

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.

- A simple greedy strategy may be: to maintain a heap with all the unused stock.
- For every element, check the smallest unused price in the heap and use it when it is profitable.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4\}$

- When $i = 1$, we put $P[1] = 4$ into the heap.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4, 3\}$

- When $i = 1$, we put $P[1] = 4$ into the heap.
- When $i = 2$, the smallest unused price is 4.
  - We cannot profit by buying at 4 and selling at $P[2] = 3$, put it in the heap.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4, 3, 1\}$

- When $i = 1$, we put $P[1] = 4$ into the heap.
- When $i = 2$, the smallest unused price is 4.
  - We cannot profit by buying at 4 and selling at $P[2] = 3$, put it in the heap.
- When $i = 3$, the smallest unused price is 3.
  - We cannot profit by buying at 3 and selling at $P[3] = 1$, put it in the heap.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4, 3, 1\}$

- When $i = 4$, the smallest unused price is 1.
  - We can profit by buying at 1 and selling at $P[4] = 6$.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4, 3\}$

- When $i = 4$, the smallest unused price is 1.
  - We can profit by buying at 1 and selling at $P[4] = 6$.
  - Profit by doing so, and update the heap.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4, 3\}$

- When $i = 5$, the smallest unused price is 3.
  - We can profit by buying at 3 and selling at $P[5] = 7$.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4\}$

- When $i = 5$, the smallest unused price is 3.
  - We can profit by buying at 3 and selling at $P[5] = 7$.
  - Profit by doing so, and update the heap.

Answer $= (6 - 1) + (7 - 3) = 5 + 4 = 9$

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Sample: $P = [4, 3, 1, 6, 7]$.
Heap: $\{4\}$

Sadly, that is actually wrong.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Consider this case: $P[4] = [1, 2, 3, 4]$.

The optimal solution is $(4 - 1) + (3 - 2) = 4$.
Our greedy solution will give us $(2 - 1) + (4 - 3) = 2$.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Crucial Observation:

- Supposed that it is optimal to buy a stock at price $P[Buy]$ and sell it at the price $P[Sell]$.
- If there exist a day i $(Buy < i < Sell)$,
  $P[Sell] - P[Buy] = (P[Sell] - P[i]) + (P[i] - P[Buy])$.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Crucial Observation:

- Supposed that it is optimal to buy a stock at price $P[Buy]$ and sell it at the price $P[Sell]$.
- If there exist a day i $(Buy < i < Sell)$,
  $P[Sell] - P[Buy] = (P[Sell] - P[i]) + (P[i] - P[Buy])$.
- The meaning of this equality is: If we buy a stock at $P[buy]$ and sell that at $P[i]$. Then, we buy a stock at $P[i]$ and sell at $P[Sell]$. It is the same as buying and selling at the optimal price. (which makes a lot of sense)

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Solution:

- We still maintain the buyable stocks with a heap.
- If we can profit by buying the lowest-priced stock in the heap and selling now, buy the stock and *sell it now*.

Problem 4

Excerpt: Given stock prices of N days $P[1..N]$. You may buy or sell one stock every day. Maximize your profit.

Solution:

- We still maintain the buyable stocks with a heap.
- If we can profit by buying the lowest-priced stock in the heap and selling now, buy the stock and *sell it now*.
- Every time we sell a stock, we give ourselves a **regret action**: buying back that stock at the price you sell it.
- No need to maintain the regret action separately, just push it in the heap so we consider it with other prices together.

Problem 4

```cpp
int main() {
    min_heap<Node> Q;
    ... // input
    for (int i = 1; i <= n; i++) {
        Q.push(p[i]); // allow yourself to buy at price p[i]
        if (!Q.empty() && Q.top().value < p[i].value) {
            // when selling at [i] will get you profit
            ans += p[i].value - Q.top().value; // update the ans with the diff
            Q.pop(); // buy at Q.top already, pop it out
            Q.push(p[i]); // allow yourself to regret selling at p[i] later
        }
    }
    ... // output
    return 0;
}
```

Problem 5

Problem Statement

Given N $(1 \leq N \leq 5 \times 10^5)$ slots to plant trees. A tree planted at slot i will give a value of $A[i]$ (can be negative).
You may not plant trees in two adjacent slots.
What is the maximum value you can get if you can plant at most K trees?

Link: P1484 - Planting Trees (Simplified Chinese)

Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

We can plant at most $\lceil \frac{N}{2} \rceil$ trees. Planting more will violate the rule.

We can greedily choose the max value in each step, then delete it and its 2 adjacent values.

- e.g. $A[1..4] = [-2, 7, 3, 4], K = 2$
- First take 7 and delete the adjacent values.
- We are left with 4, take it too and done!

Right?

Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

We can greedily choose the max value in each step, then delete it and its 2 adjacent values.

Right? **No**

- At this point, you should have realized that solutions in this lecture without a regret strategy are usually a trap.
- Coming up with a countercase to disprove your idea is a useful skill in contests though.

Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

We can greedily choose the max value in each step, then delete it and its 2 adjacent values.

Right? **No**

- e.g. $A[1..4] = [6, 7, 6, 4], K = 2$
- Instead of choosing 7, choosing both 6s will give you a better answer.

Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

We can greedily choose the max value in each step, then delete it and its 2 adjacent values.

Right? **No**

- e.g. $A[1..4] = [6, 7, 6, 4], K = 2$
- Instead of choosing 7, choosing both 6s will give you a better answer.

Problem 5

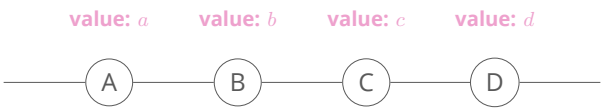
Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

To approach this problem with regrettable greedy, the difficult part is to set up a regretting strategy.

Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

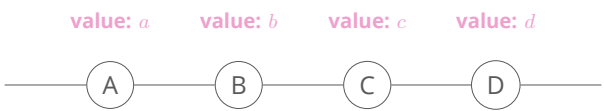
Suppose we have 4 slots here: A, B, C, and D with their respective values. The nodes are connected with a doubly linked list.



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

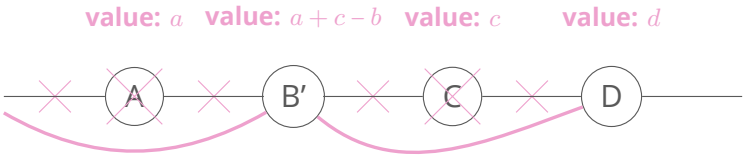
If we first chose node B, we cannot choose A and C.
Simply deleting them is incorrect (shown in the previous example), since choosing A and C instead of B may be better.



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

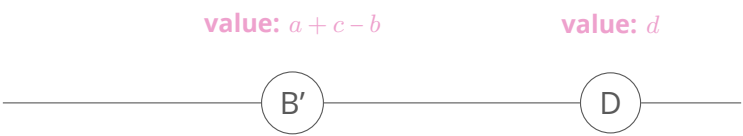
So we make a new node B', with its value being $a + c - b$, and use it to replace B. We also connect B' with the AC's neighbours.



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

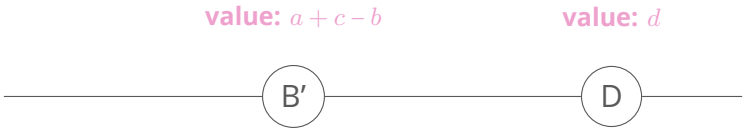
The linked list after the modification is essentially this:



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

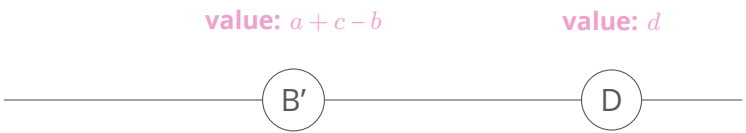
If the node we chose at the next step is B', it means that we actually **regret choosing** B but choose AC instead.



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

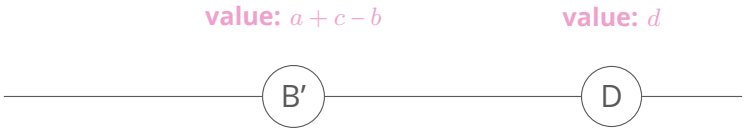
- With a doubly linked list, we can change the wiring between the nodes in $O(1)$.
- To get the largest node every step and to delete nodes. A max heap can be used with $O(\log N)$.



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

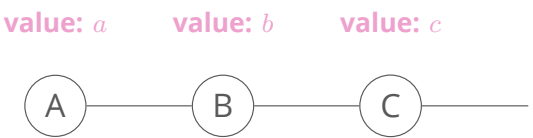
You can do most K moves, but if at some point, the largest node value is negative, you may break there.



Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

Corner case: What happens if the largest node is on the leftmost or rightmost? (e.g. a in this case)

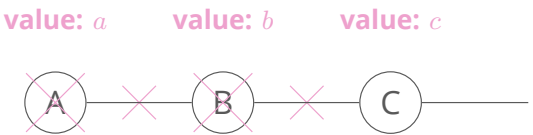


Problem 5

Excerpt: Given N slots to plant trees with different values $A[1..N]$. Maximize value gained if you can plant at most K trees that are not adjacent.

We can just simply delete node A and B without setting up a *regret node*.

If we choose A at some step, it means that $a > b$. We will never regret picking A and pick B in this scenario (A is a overall better choice).



Problem 5.2

The tree planting problem is actually a standard model for regrettable greedy. Many tasks could be reduced to it.

If slot 1 and slot N are considered adjacent now, how to modify our approach?

Problem 5.2

The tree planting problem is actually a standard model for regrettable greedy. Many tasks could be reduced to it.

If slot 1 and slot N are considered adjacent now, how to modify our approach?

- Simply transformed to using a cyclic linked list. Initialize $l[1] = N$ and $r[N] = 1$, instead of pointing them to null.

Problem 5.2

The tree planting problem is actually a standard model for regrettable greedy. Many tasks could be reduced to it.

If slot 1 and slot N are considered adjacent now, how to modify our approach?

- Simply transformed to using a cyclic linked list. Initialize $l[1] = N$ and $r[N] = 1$, instead of pointing them to null.
- The cyclic variant is actually easier as we do not need to handle corner case!
- We can actually transform chain version to cyclic version by adding a dummy node with -inf value to connect the two ends.

Link: P1792 - Planting Trees (National Training team ver.) (Simplified Chinese)

Table of Contents

Problem 5 (Revisit)

Back to the original question: how would you approach it when you do not know about regrettable greedy?

Problem 5 (Revisit)

Back to the original question: how would you approach it when you do not know about regrettable greedy?

DP, probably.

Problem 5 (Revisit)

Back to the original question: how would you approach it when you do not know about regrettable greedy?

DP, probably.

- It is easy to set up an $O(NK)$ DP.
- $dp[i][j]$: maximum profit when planting j trees at slot $1..i$
- Transition: $dp[i][j] = max(dp[i-1][j], dp[i-2][j-1] + A[i])$
- Answer: $dp[N][K]$

Problem 5 (Revisit)

Back to the original question: how would you approach it when you do not know about regrettable greedy?

DP, probably.

- It is easy to set up an $O(NK)$ DP.
- $dp[i][j]$: maximum profit when planting j trees at slot $1..i$
- Transition: $dp[i][j] = max(dp[i-1][j], dp[i-2][j-1] + A[i])$
- Answer: $dp[N][K]$

But it is too slow to do so. What can you do to speed it up?

Problem 5 (Revisit)

Think of an easier question first: if there are no constraints on how many trees you can plant, how do you do it with DP?

Problem 5 (Revisit)

Think of an easier question first: if there are no constraints on how many trees you can plant, how do you do it with DP?

- $dp[i][0/1]$: maximum profit when planting any number of trees at slot $1..i$, the second state denotes choosing to plant tree on slot i or not.
- Transition: $dp[i][0] = max(dp[i-1][0], dp[i-1][1])$
  $dp[i][1] = dp[i-1][0] + A[i]$
- Answer: $max(dp[N][0], dp[N][1])$

That is an $O(N)$ DP, which runs quick enough.

Problem 5 (Revisit)

Do you recall any tricks that turn a *constrained problem* into an *unconstrained problem*? Let us also try to think about the graph if we plot ans against K.

I hope it is obvious enough that ans is a convex function depending on K. Since the profit you gain from planting more trees should be decreasing.

Problem 5 (Revisit)

Do you recall any tricks that turn a *constrained problem* into an *unconstrained problem*? Let us also try to think about the graph if we plot ans against K.

I hope it is obvious enough that ans is a convex function depending on K. Since the profit you gain from planting more trees should be decreasing.

What does it mean?

Problem 5 (Revisit)

Do you recall any tricks that turn a *constrained problem* into an *unconstrained problem*? Let us also try to think about the graph if we plot ans against K.

I hope it is obvious enough that ans is a convex function depending on K. Since the profit you gain from planting more trees should be decreasing.

What does it mean? **ALIEN TRICKS**!

Problem 5 (Revisit)

What does it mean? **ALIEN TRICKS**!
We can add a cost, λ, to every tree planted.

- $dp[i][0/1]$: maximum profit when planting any number of trees at slot $1..i$, the second state denotes choosing to plant tree on slot i or not.
- Transition: $dp[i][0] = max(dp[i-1][0], dp[i-1][1])$
  $dp[i][1] = dp[i-1][0] + A[i] - \lambda$
- Answer: $max(dp[N][0], dp[N][1])$

Binary search on the λ to make it plant at most K trees with a certain λ.
(Details omitted)

Time Complexity: $O(N \log C)$

Why?

The question worth asking here is: why do both Regrettable Greedy and Alien Tricks can solve this problem? Coincidence?

What is in common between these two techniques?

Why?

The question worth asking here is: why do both Regrettable Greedy and Alien Tricks can solve this problem? Coincidence?

What is in common between these two techniques? It turns out that we have to find common ground with a third technique.

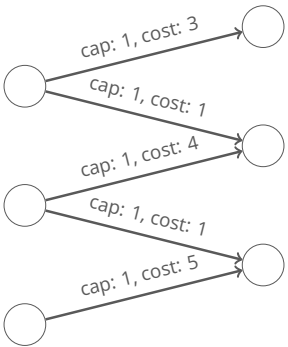
Problem 5 (Re-revisit)

Back to the original question: how would you approach it when you do not know about regrettable greedy? (and you do not know about alien tricks)

(and maybe you are good at Flow?)

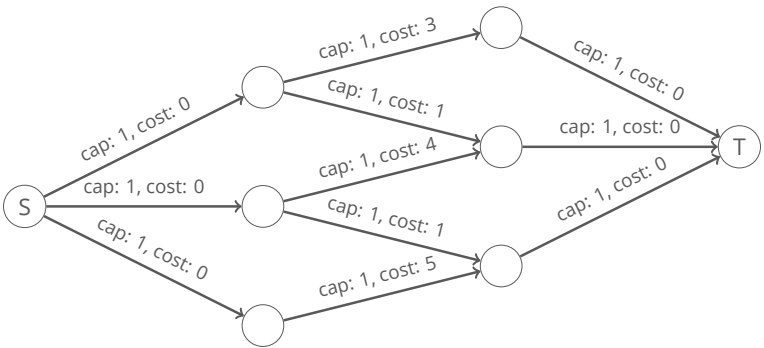
Problem 5 (Re-revisit)

e.g. $A[1..5] = [3, 1, 4, 1, 5]$, the flow model is as follows:



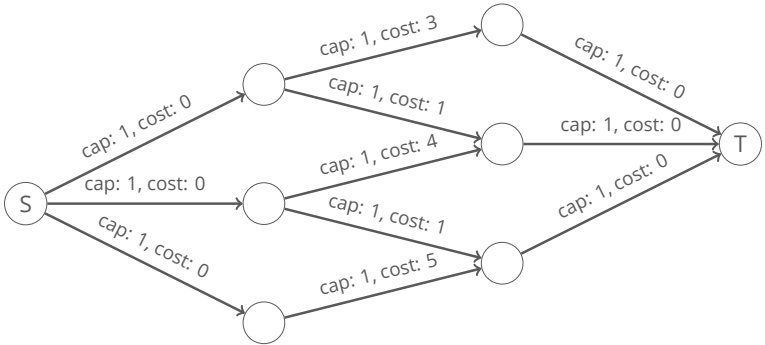
Problem 5 (Re-revisit)

e.g. $A[1..5] = [3, 1, 4, 1, 5]$, the flow model is as follows:



Problem 5 (Re-revisit)

Limiting the flow from source to be at most K. Then the answer is equal to running max-cost flow with Edmond-Karp (take negative for cost to find max-cost).



Problem 5 (Re-revisit)

Now we have 3 solutions! (Although flow is very slow in comparison) What does Flow have that is related to Alien Tricks and Regrettable Greedy?

Problem 5 (Re-revisit)

Now we have 3 solutions! (Although flow is very slow in comparison) What does Flow have that is related to Alien Tricks and Regrettable Greedy?

Alien Tricks:

- According to Edmond-Karp algorithm, the length of the augmented path found is **monotonically increasing**.
- The cost is a **convex function** on the amount of flow.

Problem 5 (Re-revisit)

Now we have 3 solutions! (Although flow is very slow in comparison) What does Flow have that is related to Alien Tricks and Regrettable Greedy?

Alien Tricks:

- According to Edmond-Karp algorithm, the length of the augmented path found is **monotonically increasing**.
- The cost is a **convex function** on the amount of flow.

Regrettable Greedy:

- In flow, you reverse the edge capacity (and cost) after flowing through it to build the residue network. That is providing a **regret action**: regret flowing through an edge.

Simulated min/max-cost flow（模擬費用流）

Running min/max-cost flow on a graph with $N \sim 10^5$ will probably lead you to TLE (it actually depends on the graph though).

- We can build the flow model to serve as a proof of concept, which helps you to understand the question deeper. e.g., the flow model actually tells you that the answer (cost) is a convex function on flow.
- For some flow models, we can use other techniques to *simulate* the Flow algorithm without running normal Flow algorithms directly. This usually includes using Alien Tricks or Regrettable Greedy.

Problem 6

Problem Statement

Given two sequences of length N ($1 \le N \le 10^6$), $A[1..N]$ and $B[1..N]$.
You have to pick K numbers in each sequence and ensure at least L positions are chosen in both sequences.
Maximize the sum of the $2K$ numbers.

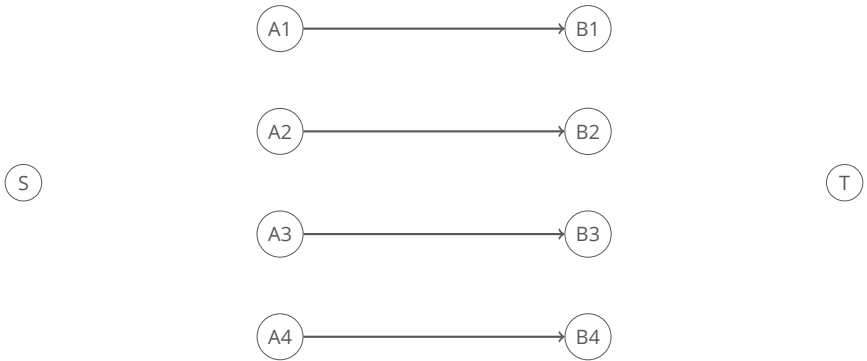
Link: NOI 2019 - Sequence

Problem 6

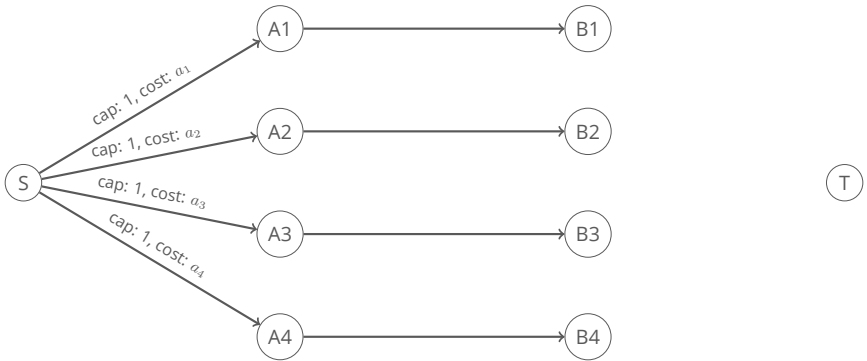
Excerpt: Given 2 sequences of length N, maximize sum when picking K numbers each and ensure at least L position is the same.

At least L position is the same, meaning that we have $K - L$ free positions. We can try to build a flow model for it.

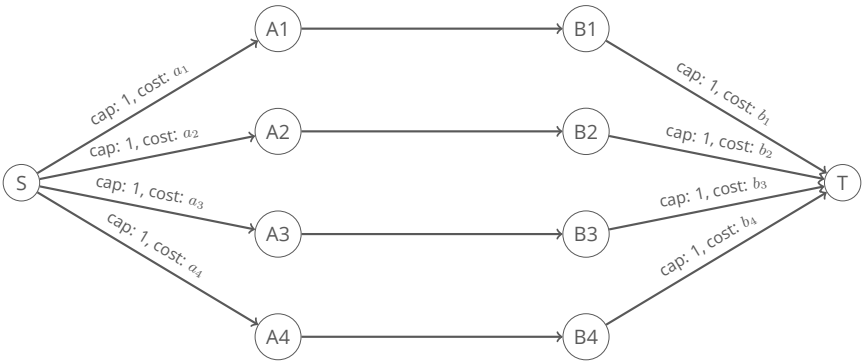
Problem 6



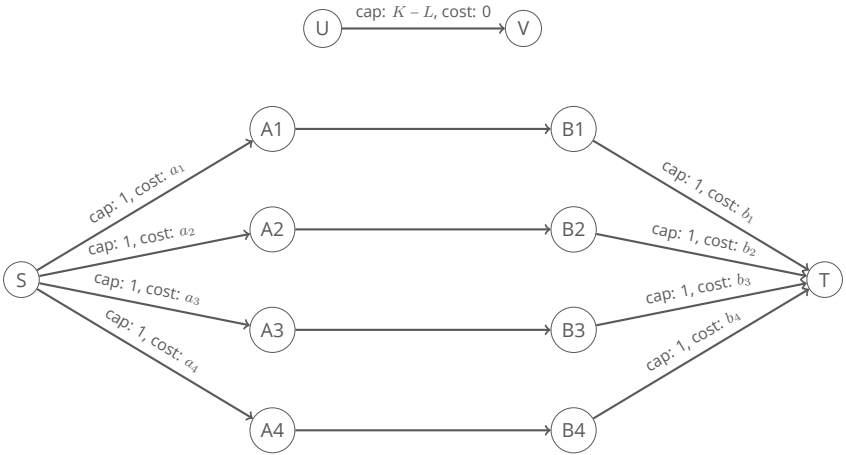
Problem 6


A1
B1
cap: 1, cost: a_1
A2
B2
cap: 1, cost: a_2
S
T
cap: 1, cost: a_3
A3
B3
cap: 1, cost: a_4
A4
B4

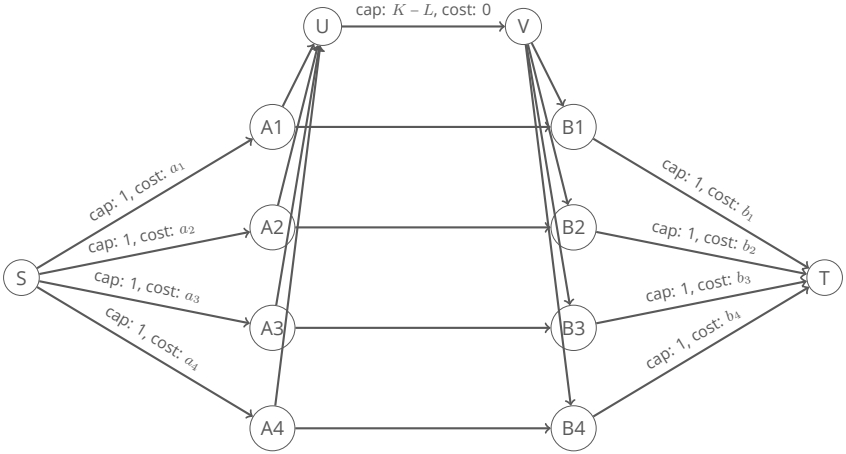
Problem 6


S
A1
A2
A3
A4
B1
B2
B3
B4
T
cap: 1, cost: a_1
cap: 1, cost: a_2
cap: 1, cost: a_3
cap: 1, cost: a_4
cap: 1, cost: b_1
cap: 1, cost: b_2
cap: 1, cost: b_3
cap: 1, cost: b_4

Problem 6


U
cap: $K - L$, cost: 0
V
S
A1
A2
A3
A4
B1
B2
B3
B4
T
cap: 1, cost: a_1
cap: 1, cost: a_2
cap: 1, cost: a_3
cap: 1, cost: a_4
cap: 1, cost: b_1
cap: 1, cost: b_2
cap: 1, cost: b_3
cap: 1, cost: b_4

Problem 6


U
V
cap: $K - L$, cost: 0
A1
A2
A3
A4
B1
B2
B3
B4
S
T
cap: 1, cost: a_1
cap: 1, cost: a_2
cap: 1, cost: a_3
cap: 1, cost: a_4
cap: 1, cost: b_1
cap: 1, cost: b_2
cap: 1, cost: b_3
cap: 1, cost: b_4

Problem 6

After building the flow model, we can limit the flow from source to be K and run max-cost flow to get the answer. It can get you partial score but not enough to get full.

Problem 6

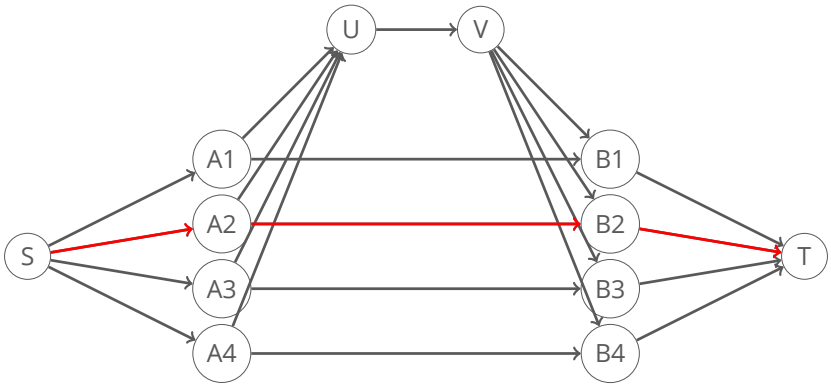
Consider simulating the process of Edmond-Karp, what kind of augmenting paths would you find?

- We will never undo flow in edges connecting to S or T (can ignore the edge after use), but the direction of the middle edges may change.

1. $S \to A_i \to B_i \to T$
2. $S \to A_i \to U \to V \to B_j \to T$
3. $S \to A_i \to B_i \to V \to U \to A_j \to B_j \to T$
4. $S \to A_i \to U \to A_j \to B_j \to T$
5. $S \to A_i \to B_i \to V \to B_j \to T$

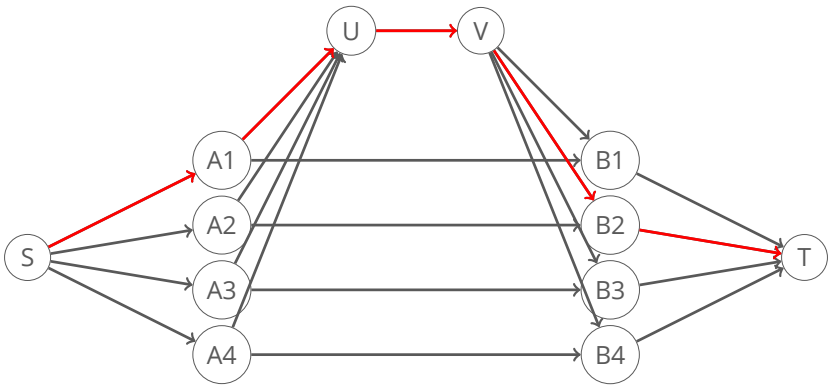
Problem 6

$S \to A_i \to B_i \to T$ (choose A_i and B_i)



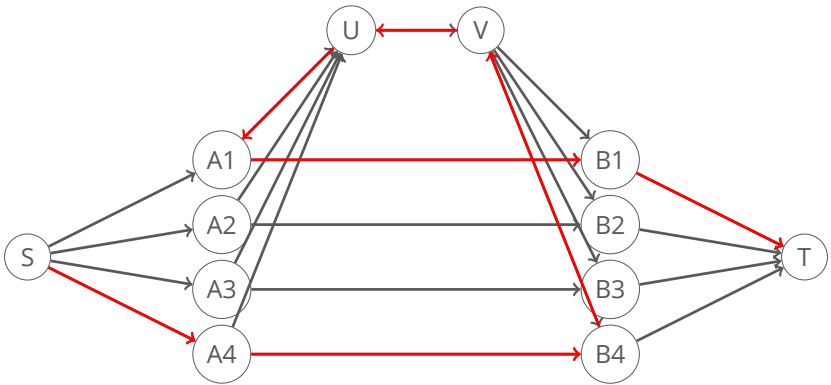
Problem 6

$S \to A_i \to U \to V \to B_j \to T$ (choose A_i and B_j, use 1 unit of free flow)



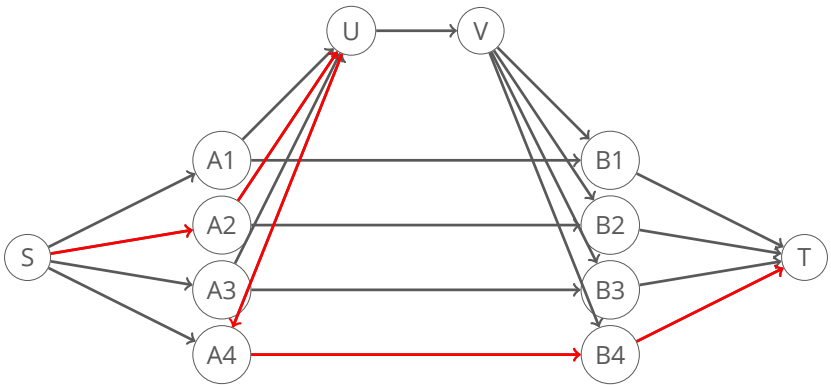
Problem 6

$S \to A_i \to B_i \to V \to U \to A_j \to B_j \to T$ (choose A_i and B_j, that A_j and B_i is chosen before, free up 1 unit of free flow)



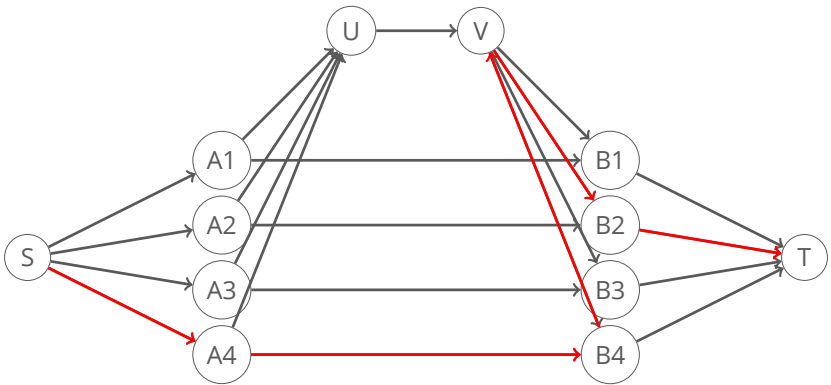
Problem 6

$S \to A_i \to U \to A_j \to B_j \to T$ (choose A_i and B_j, that A_j is chosen before)



Problem 6

$S \to A_i \to B_i \to V \to B_j \to T$ (choose A_i and B_j, that B_i is chosen before)



Problem 6

There exists a more complicated augmented path but could be reduced to these 5.

We can use max-heap to maintain the value used for these 5 augmented paths.

Q: $\{a_i + b_i\}$ where both a_i and b_i have not been chose

Q_a: $\{a_i\}$ where a_i has not been chose

Q_b: $\{b_i\}$ where b_i has not been chose

Q'_a: $\{a_i\}$ where a_i have not been chose but b_i has been chose

Q'_b: $\{b_i\}$ where b_i have not been chose but a_i has been chose

Problem 6

Hence, the 5 augmented paths are:

(we start with $K - L$ free flow)

1. $Q.top()$
2. $Q_a.top() + Q_b.top()$ (available when there are free flow)
3. $Q'_a.top() + Q'_b.top()$ (add 1 to free flow)
4. $Q'_a.top() + Q_b.top()$
5. $Q_a.top() + Q'_b.top()$

Each step get the maximum from these 5 expressions, and do proper tie-breaking, then done!

Conclusion

1. Regrettable Greedy is not commonly seen in OI. However, it could sometimes help you to solve tricky ad-hoc tasks in an unintended way.
2. Try to understand the nature of the algorithm and tricks you have learnt. What is the limitation of them? In what scenario could they be applied?
3. Even if you cannot fully understand it, try to learn how to use it! Do not place yourself in a disadvantageous position against hardworking contestants worldwide.

Practice Question

- CF436E - Cardboard Box
- CF730I - Olympiad in Programming and Sports
- NEERC2016 - Mole Tunnels
- P4694 - [PA2013] Raper (Simplified Chinese)
- NOI 2017 - Vegetables (Simplified Chinese)
- ICPC World Finals 2018 C - Conquer the World

References

- 【蒟蒻算法】反悔貪心 - jvruo (Main reference, recommend to check it out) (Simplified Chinese)
- 反悔貪心 - nth_element (Simplified Chinese)
- 模擬費用流小記 - command_block (Simplified Chinese)
- NOI 2019 Sequences Solution - command_block (Simplified Chinese)