CS101 Final Review

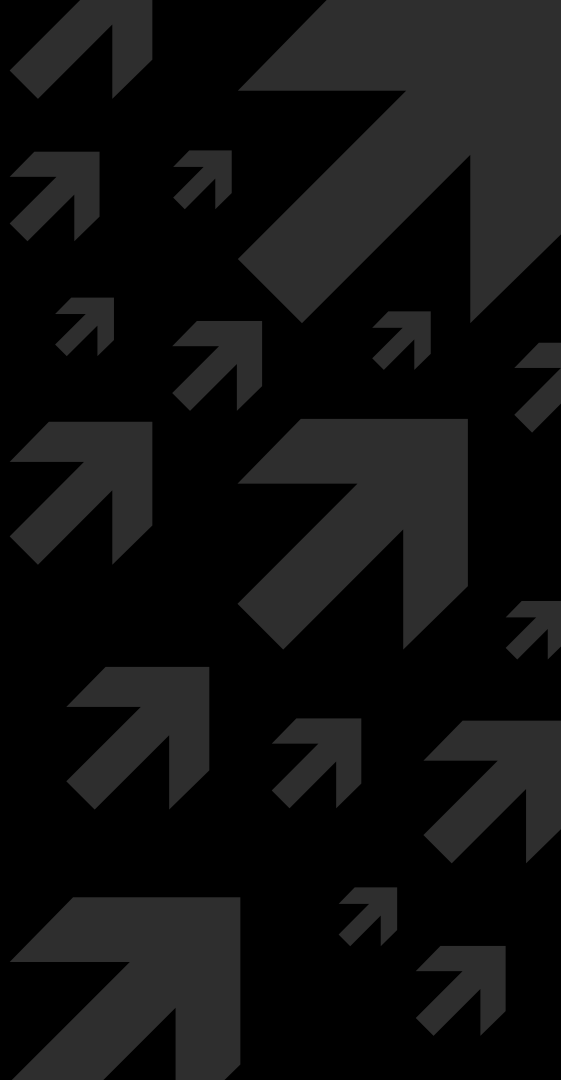
DAG, Topological Sort, Greedy Algorithms

Final

- **Logistics**
  - **Time:** 12/28 Wednesday 8:00 AM ~ 10:00 AM
  - **Exam Classroom:** Online (details TBD)
- **Content**
  - 10 × True or False + 5 × Single Choice + 5 × Multiple Choices
  - 4 questions covering Graph algorithms, Greedy, DP, NP-Complete…

1. Topological Sort

Definition/Algorithm/Notes



Toposort - Definition

- **D**irected **A**cyclic **G**raph
- Tasks (vertices) with dependencies (edges)
  - Task A must be done before task B if …
  - **Vertex A appears before vertex B** in topological sorting if …
  - … if there exists a **path A → … → B**

Toposort - Algorithm

```
TopologicalSort(G):
    Q ← empty queue
    T ← empty list to store topological sorting
    In ← list to store in-degree of vertices
    Push vertices with 0 in-degree into Q
    While Q is not empty :
        Pop vertex u from Q
        Add u to T
        For each neighbor v of u in G :
            In[v] ← In[v] - 1
            if In[v] = 0 :
                push v into Q
    Return T
```

Toposort - Notes

- Can **detect cycle** in directed graphs!
- **Time complexity**: $\Theta(|V| + |E|)$
- This algorithm returns one possible topological sorting of G, but …
- … but G might have different topological sorting**s**!

Toposort - Notes

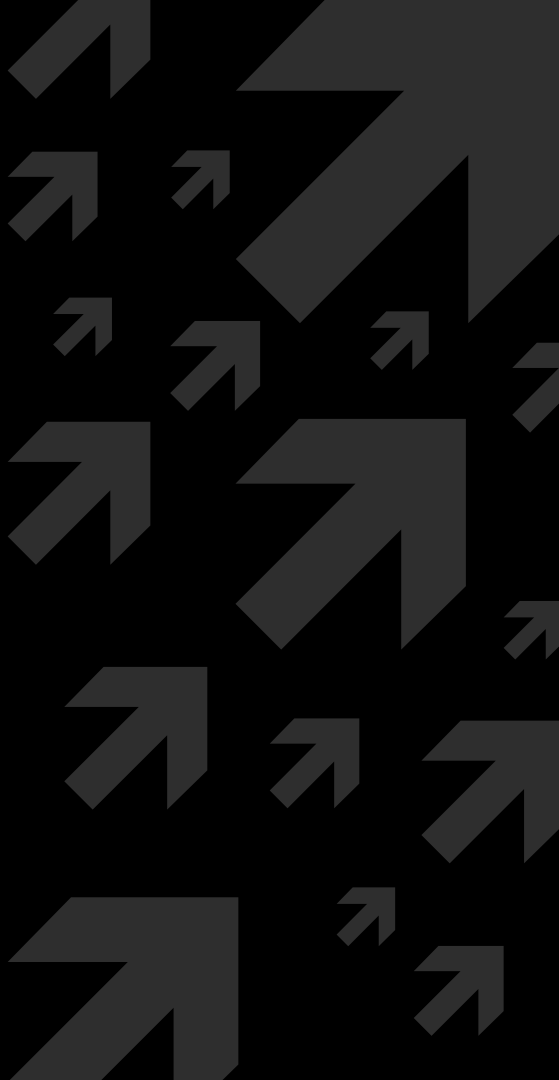
➔ Can **detect cycle** in directed graphs!

```
TopologicalSort(G):
    Q ← empty queue
    T ← empty list to store topological sorting
    In ← list to store in-degree of vertices
    Push vertices with 0 in-degree into Q
    While Q is not empty :
        Pop vertex u from Q
        Add u to T
        For each neighbor v of u in G :
            In[v] ← In[v] - 1
            if In[v] = 0 :
                push v into Q
    Return T
```

If there exists some vertex `u` with `In[u] > 0`:
There exists a cycle!

2.
Greedy Algorithm

Algorithm/Exchange Argument/Examples



Greedy Algorithm

- **What is greedy algorithm?**
  - Focus on one particular partial solution and we attempt to extend that solution
  - Sometimes the partial solutions should lead to a feasible solution which is also optimal

Greedy Algorithm - Examples

- **What is a greedy algorithm like?**
  - **Making Coin Change:** sort and from large ¥100 to small ¥1
  - **Interval Scheduling:** sort by earliest-finish-time-first
  - **Minimizing Lateness:** sort by earliest-deadline-first

Greedy Algorithm - Proof

- **How to prove a greedy algorithm is optimal?**
  - By **Exchange Argument**!
  - **Step1 Two Solutions:** Label greedy solution and an optimal solution.
  - **Step2 Observation:** Compare these two solutions.
  - **Step3 Exchange** (optimal solution → greedy solution) and show that things become no worse.

Greedy Algorithm - Proof

Supplementary material

Step 1: Label your algorithm's solution, and a general solution. For example, let $A = \{a_1, a_2, \ldots, a_k\}$ be the solution generated by your algorithm, and let $O = \{o_1, o_2, \ldots, o_m\}$ be an arbitrary (or optimal) feasible solution.

Step 2: Compare greedy with other solution. Assume that your arbitrary/optimal solution is not the same as your greedy solution (since otherwise, you are done). Typically, you can isolate a simple example of this difference, such as one of the following:

- there is an element of O that is not in A and an element of A that is not in O, or
- there are 2 consecutive elements in O in a different order than they are in A (i.e. there is an inversion).

Step 3: Exchange. Swap the elements in question in O (either swap one element out and another in for the first case, or swap the order of the elements in the second case), and argue that you have a solution that is no worse than before. Then argue that if you continue swapping, you can eliminate all differences between O and A in a polynomial number of steps without worsening the quality of the solution. Thus, the greedy solution produced is just as good as any optimal (or arbitrary) solution, and hence is optimal itself.

Greedy Algorithm - Proof

Supplementary material

Comments

- Be careful about using proofs by contradiction starting with the assumption $G \neq O$. Just because your greedy solution is not equal to the selected optimal solution does not mean that greedy is not optimal – there could be many optimal solutions, and your greedy one just isn't the optimal solution you selected. So assuming $G \neq O$ may not get you any contradiction at all, even if greedy works.
- You need to argue why the 2 elements you're swapping even exist out of order, or exist in O but not in A, etc.
- Remember you need to argue that *multiple* swaps can get you from your selected solution to greedy, as one single swap will usually not suffice. Also, make sure that any step you make (and not just the first one) doesn't hurt the solution quality.

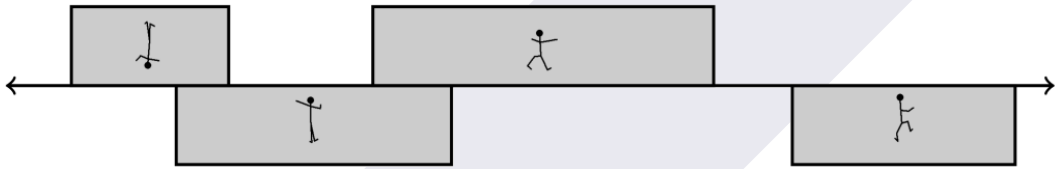
Exchange Argument - Example

5. (10 points) Placing Bus Stations

A city is interested in designing a new bus line that runs on a straight road. However, they are unsure of where to place the bus stops. The city surveyed bus users along the route for (a) where they live and (b) how far they would be willing to walk to a bus station. Design a greedy algorithm that minimizes the number of bus stops required to cover all surveyed bus users. Give an exchange argument to prove that your algorithm is correct and analyse the run time complexity.

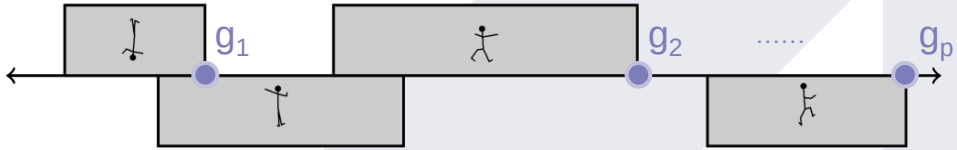
Assume the survey included n users with user i living at x_i and willing to walk a distance d_i(in either direction). You may assume that all elementary arithmetic operations take unit time.

For example, the following picture shows 4 users and the ranges they are willing to walk for a bus. 3 stations are needed at least.



Exchange Argument - Example

- **Step1:** Label greedy solution and an optimal solution
  - Greedy solution:

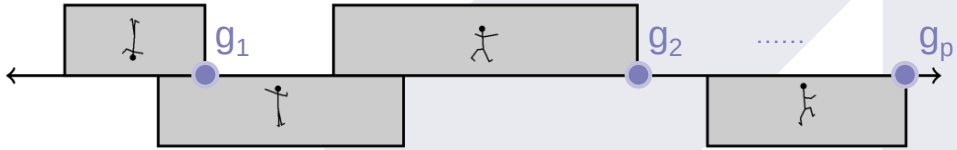


  - Optimal solution:

$q \leq p$

Exchange Argument - Example

- **Step1:** Label greedy solution and an optimal solution
  - Greedy solution:



  - Optimal solution:

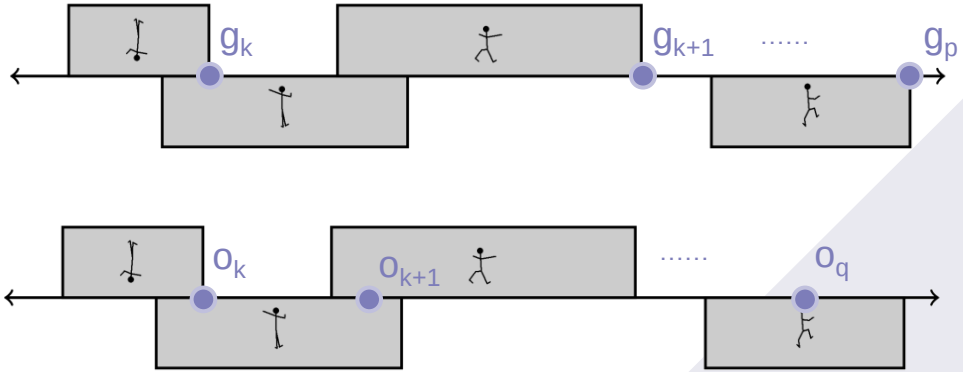
$q \le p$

Case1: G=O

Case2: G≠O Assume $g_1=o_1$, $g_2=o_2$, …, $g_k=o_k$ for the largest possible value of k

Exchange Argument - Example

➔ **Step2 Observation:** Compare these two solutions

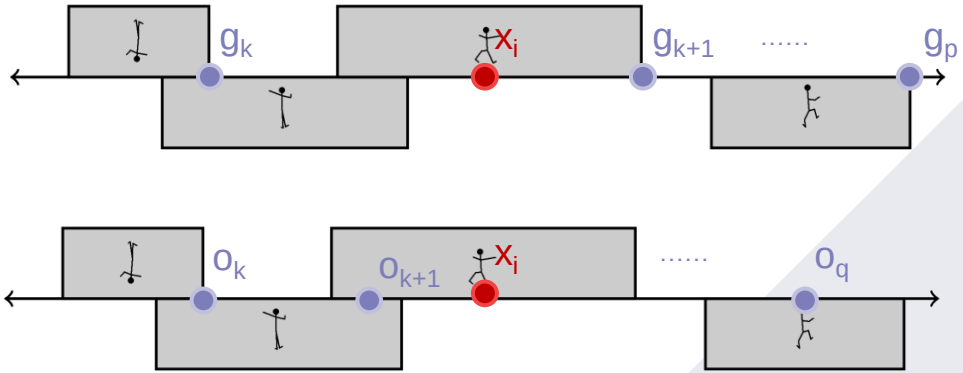


$g_{k+1} \geq o_{k+1}$

Why?

Exchange Argument - Example

➔ **Step2 Observation:** Compare these two solutions



$$g_{k+1} \geq o_{k+1}$$

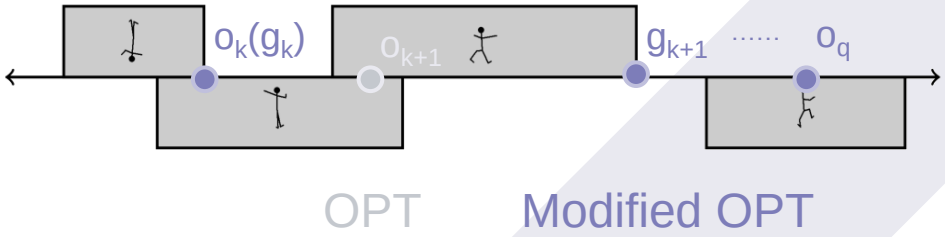
Let x_i be the next user uncovered by k

$x_i - d_i \leq o_{k+1} \leq x_i + d_i = g_{k+1}$

x_i should be covered by greedy algorithm

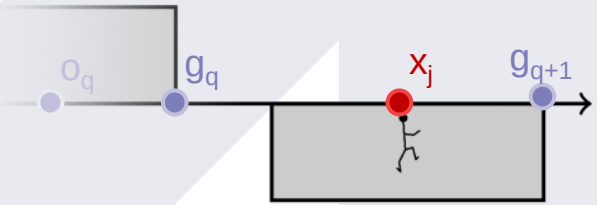
Exchange Argument - Example

- **Step3 Exchange** (OPT→ Greedy)
  - We can replace o_{k+1} with g_{k+1}
  - Since previous users are already covered by 1 … k
  - ... and g_{k+1} also cover the next uncovered user



Exchange Argument - Example

- **Step3 Exchange** (Not worsen OPT)
  - $o_{k+1} \rightarrow g_{k+1}$, $o_{k+2} \rightarrow g_{k+2}$, …, $o_{q-1} \rightarrow g_{q-1}$
  - Prove p≤q by contradiction: suppose p>q
  - By greedy algorithm, there exists some x_j such that $o_q \leq g_q < x_j - d_j$
  - … which contradicts the fact that OPT should cover x_j
  - p≤q and q≤p implies p=q i.e. Greedy is optimal.



Exchange Argument - Example

4. (9 points)

Given a set of $n \geqslant 3$ distinct positive numbers $S = \{s_1, s_2, \cdots, s_n\}$, we want to find a permutation $A = \langle A_1, \cdots, A_n \rangle$ of S, where $A_i \in S$ for all $i \in \{1, \cdots, n\}$, such that

$$f(A) = A_1^2 + \sum_{i=2}^{n} (A_i - A_{i-1})^2$$

is maximized.

(a) (3') Describe your algorithm that finds the permutation A for which $f(A)$ is maximized. Use **pseudocode** or **natural language**.

(b) (4') Prove the correctness of your algorithm by showing that your choice on the value of A_1 is optimal, i.e. any other choice would not lead to a better solution.

(c) (2') Time complexity. Your algorithm should be $O(n \log n)$.

Exchange Argument - Example

- **Step1:** Label greedy solution and an optimal solution
  - Greedy solution: $G = \{s_n, s_1, s_{n-1}, s_2, \dots, G_{k+1}, \dots\}$
  - Optimal solution: $O = \{s_n, s_1, s_{n-1}, s_2, \dots, O_{k+1}, \dots\}$

$G_1=O_1=s_n$, $G_2=O_2=s_1$, …, $G_k=O_k$ for the largest possible value of k

Exchange Argument - Example

- **Step2 Observation:** Compare these two solutions
  - W.L.O.G consider k is even, then by greedy algorithm:
    - G_{k+1} is the largest among $G_{k+1}, \ldots, G_n$
    - G_k is the smallest among $G_k, G_{k+1}, \ldots, G_n$

    proof of odd k is similar
  - Greedy solution: $G = \{s_n, s_1, \ldots, G_{k+1}, \ldots\}$
  - Optimal solution: $O = \{s_n, s_1, \ldots, O_{k+1}, \ldots, G_{k+1}, \ldots\}$
  - Key Observation: $O_{k+1} \leq G_{k+1}$ by greedy algorithm

Exchange Argument - Example

- **Step3 Exchange** (OPT→ Greedy)
  - Reverse O_{k+1}, …, G_{k+1} in O to eliminate difference at k+1
  - OPT: $O = \{s_n, s_1, \ldots, O_{k+1}, \ldots, G_{k+1}, \ldots\}$
  - Modified OPT: $O' = \{s_n, s_1, \ldots, G_{k+1}, \ldots, O_{k+1}, \ldots\}$
  - Not worsen OPT: $f(O') - f(O) \geq 0$ details are not shown here
  - Apply exchange to k+1, k+2, …, n → Greedy is optimal

Thanks!

Any questions?

Good Luck!

Contact me via lianyh@shanghaitech.edu.cn if you have any doubt