Road Service
JEREMY CHOW 30/6/2018
Problem Statement

- JOI 2018 Spring Camp Day 2 Problem B

- Given a tree with N vertices
- Add K new edges to the graph so that the total distance is minimized

- Total distance = sum of shortest paths between every two points
Problem Statement

- $N = 1000$
- **Output only task**

- You are **NOT** required to find out the optimal answer
- Nearly-optimal is good enough
Which kind of graph have small pairwise distance?
- Chain
- Total distance = (1+2+3)+(1+2)+1=10
Which kind of graph have small pairwise distance?
- Star
- Total distance = $(1+2+2)+(1+2)+1=9$
Direction

- Which kind of graph have small pairwise distance?
- Star or Chain?

- Actually, Chain graph’s total pairwise distance is $O(N^3)$
- But Star graph’s total pairwise distance is only $O(N^2)$

- Difference between those two type graph will get bigger when N is large
Direction

- So, if we concentrate all k extra edges to one particular vertex
- Make the graph like a star graph
- We can get graph that gives good result
Solution ~6 points

- Randomly output K edges

- You should at least get this 6 points.
Solution ~30 points

- Output 1-2, 1-3, 1-4, ..., 1-k+1
- Turn the graph into star shape
- Easy 30 points 😊
Solution ~30 points

- You may try exhaust the center
- Trying x-1, x-2, x-3, ... x-k

- Output the best solution
- But it won’t improve lots
Solution ~60 points

- Exhaust the center of the star

- But which vertices should we choose to connect with the center?
  - Random is obviously not good

- We want to make every vertex as close to the center as possible at the end
  - So that the total pairwise distance is most likely very small
One way is to connect to the vertices having large degree.

Larger degree -> most likely more vertices will closer to center.

Good ways to connect the vertices.

You may use another “good ways” to connect the vertices.
Solution ~60 points

- Try making every vertex as center
- Connect center with k vertices having large degree
- Compare the total pairwise distance with your current best solution
- Output the best solution

- You will get 60 points if you implement it correctly
Solution ~60 points

- How to calculate the total pairwise distance?
- Do BFS on every vertex
- $O(N^2)$

- Notice that you should avoid adding edge that is already exist in the original graph
- Also avoid adding self loop
Let the set of vertices connecting to the center be $X$

Actually connecting center to the vertices having large degree will give us a pretty good initial state

We will try to improve our solution starting from that

- Hill climbing
Solution ~100 points

- Let each step be the following:
  - Remove one node from X and add one node to X
  - i.e. Change one of the edge connecting to the center

- There are $O(NK)$ possible steps you can choose at the same time
  - $O(K)$ vertices you want to remove, $O(N)$ vertices you want to add
Solution ~100 points

- You may try all possible steps and choose the best step
- Repeat this process until you can’t make any improvement
However, we find that the evaluation function (total pairwise distance) takes too much time to calculate.

- $O(N^2)$

We can replace the evaluation function by following:

- Sum of distance from the center to other vertices
- $O(N)$
Solution ~100 points

- Also, you may not try to improve all the initial solutions with different center

- You can just improve solutions having top 10 initial score (evaluated by the function motioned before)

- Help the program to run faster
Solution ~100 points

1. Calculate the initial solution with center $i$ ($1 \leq i \leq n$)

2. For the initial solution having top 10 initial score:
   - Try every possible step (remove and add 1 node)
   - Choose the best step and update the solution
   - Repeat this process until you can’t make any improvement more
   - Update the answer

- With this algorithm, you will get 90~100 points
Solution ~100 points

- If you replace “connect the center to the vertex with large degree” with “connect the center to the vertex such that the total distance reduce the most” and repeat it for K times

- You will get ~90 points even without the improvement (Hill climbing)
Solution ~100 points

- If you greedy select \((k+1)\) “good” vertex first, then exhaust the center

- You may also get ~90 points
Solution ~100 points

- If you think the full solution algorithm run too slow
- You may limit the number of times to make improvement
- Also, you may not trying all possible step and using some randomization
  - Randomly remove one node and add the best node to X
  - Each improvement takes $O(N)$
- You will get ~80 points if you do that
Solution ~100 points

- Also notice that test case 1 constraint is small

- You may find the optimal solution by exhaustion
  - Or with hand (if you are smart enough)
Techniques

- Output only task
- Can start working after looking the input data
- Visualizing the input may often helps
- You can even solve some input by hand (Test data 1)

- Do not require a rigorous solution
- Well designed greedy or DP algorithm can get you high score
- You can also try to improve your solution by hill climbing algorithm after determining the initial state
Techniques

- Try to make good use of your 5 hours contest time
- Do not spend all the times on it
- Try to find easy and short solutions first
- Get some nice points as fast as possible

- Try to design an algorithm that can be rerunning after a small change of parameter
Techniques

- There is no running time limit for your program
- You may just let it run a long time locally
- The official solution also takes a long time to output 1 test case

- When the program is running, you can move on to the next tasks
- Just make sure your program will search for some “good” solution
- It feels bad when you only get few points after running the program for 1 hour
Little patterns, big canvas

solution

Lau Chi Yung

2018/07/2
Subtask 1

\[ p_i = i - 1 \]

- One linear chain from 1 to \( N \)
- The direction of the arrows does not matter
- Amber can draw simultaneously on any adjacent nodes
Subtask 1

- No matter what, when pattern 1 is being drawn, pattern 2 should be drawn simultaneously.
- Better draw pattern 1 before drawing anything else, so that pattern 2 will have fewer strokes left.
- Strategy: greedily draw from left to right.
- Time complexity: $O(N)$

---

right hand has nothing to do
Subtask 2

\[ N \leq 1000, s_j = 1 \]

- Model patterns as vertices, and partner patterns as parent
- We get a forest (not only one single tree)
- Each pattern only consists of one stroke
- Again, direction of arrows does not matter
Subtask 2

- We can never draw patterns on different trees simultaneously
  \[ \Rightarrow \text{we can deal with each tree one by one} \]
- To ease our implementation, let tree roots be all nodes \( r \)
  where \( p_r = 0 \)
Subtask 2

- Always prefer drawing two patterns simultaneously
  ⇒ want to find the largest number of edges such that all vertices in these edges are distinct
- i.e. want to find the maximum matching
Subtask 2

- Always prefer drawing two patterns simultaneously
  \[ \Rightarrow \text{want to find the largest number of edges such that all vertices in these edges are distinct} \]
- i.e. want to find the maximum matching
- answer \( = N - |\text{maximum matching}| \)
- answer \( = 7 - 3 = 4 \)
Subtask 2

- Always prefer drawing two patterns simultaneously
  \[ \Rightarrow \text{want to find the largest number of edges such that all vertices in these edges are distinct} \]
- i.e. want to find the maximum matching
- answer = \( N - |\text{maximum matching}| \)
- answer = 7 - 3 = 4

maximum matching may not be unique
Subtask 2

- How to find maximum matching?
- Tree is a bipartite graph
  - simply divide the tree into odd level nodes and even level nodes
- Bipartite matching - *Hopcroft-Karp algorithm*
- Underlying mechanism is same as *Dinic’s algorithm*
- Time complexity: \( O(E\sqrt{V}) = O(V\sqrt{V}) \)
  - *for trees, \( E = V - 1 \)

*Figure: Bipartite graph*
Subtask 3

$s_j = 1$

- Same as subtask 2, want to find maximum matching
- Maximum matching on trees can be solved with Dynamic Programming
- $DP_0(u) = |\text{maximum matching}|$ in the subtree rooted at $u$ without matching $u$
- $DP_1(u) = |\text{maximum matching}|$ in the subtree rooted at $u$ with or without matching $u$

\[
DP_0(u) = \sum_{v \in \text{children of } u} DP_1(v)
\]

\[
DP_1(u) = \begin{cases} 
DP_0(u) + 1 & \text{if some } v \text{ is not matched} \\
DP_0(u) & \text{otherwise}
\end{cases}
\]

\[
\text{answer} = \sum_{p_r=0} DP_1(r)
\]

Time complexity: $O(N)$
Subtask 3

For example

\[ DP_0(a) = 1, \quad DP_1(a) = 1 \]
Subtask 3

For example

- $DP_0(a) = 1$, $DP_1(a) = 1$
- $DP_0(b) = 0$, $DP_1(b) = 1$
Subtask 3

For example

- $DP_0(a) = 1, \; DP_1(a) = 1$
- $DP_0(b) = 0, \; DP_1(b) = 1$
- $DP_0(c) = 0, \; DP_1(c) = 0$
Subtask 3

For example

- $DP_0(a) = 1$, $DP_1(a) = 1$
- $DP_0(b) = 0$, $DP_1(b) = 1$
- $DP_0(c) = 0$, $DP_1(c) = 0$
- $DP_0(r) = 1 + 1 + 0 = 2$
Subtask 3

For example

- $DP_0(a) = 1, \; DP_1(a) = 1$
- $DP_0(b) = 0, \; DP_1(b) = 1$
- $DP_0(c) = 0, \; DP_1(c) = 0$
- $DP_0(r) = 1 + 1 + 0 = 2$
- $DP_1(r) = DP_0(r) + 1 = 3$
Subtask 4

\[ N \leq 1000 \]

- Designed for solutions aiming to solve subtask 5 that are not efficient enough
Subtask 5

- No matter what, when a leaf is being drawn, its parent should be drawn simultaneously.
- Better draw the leaves before drawing anything else, so that their parents will have fewer strokes left.
Subtask 5

- Repeatedly draw on an existing leaf and its parent
- When a node is completed, remove that node
Subtask 5

- Implement with DFS; or
- perform a topological sort, and then linear scan
  - *topological sort* sorts vertices of a directed acyclic graph (DAG) in a way that the parent of a vertex comes before itself
  - a rooted tree with edges directed from children to parent forms a DAG
  - *topological sort* is implemented as DFS
- Time complexity: $O(N)$
Osu!

Charlie Li 2018/07/02
Problem statement

• Given a sequence of N queries of the following type
  • Add a point \((S, T)\) to set A or set B
  • Query \(\min(S_A \times S_B + T_A + T_B)\) where \((S_A, T_A) \in A\) and \((S_B, T_B) \in B\)

• Initially, there is a plan \((A, 0)\) in A and a plan \((B, 0)\) in B.

• Constraints:
  • \(1 \leq N \leq 500000\), \(1 \leq S_i, T_i \leq 10^9\)
Subtasks

• Subtask 1 (11 points): N <= 500
• Subtask 2 (21 points): N <= 5000
• Subtask 3 (15 points): N <= 50000
• Subtask 4 (12 points): N <= 100000
• Subtask 5 (17 points): $S_i, T_i <= 10000$
• Subtask 6 (24 points): No additional constraints
Score Distribution by Subtask

Subtask

no. of participants

1 2 3 4 5 6
Score Distribution by Participant

- Score 100: 1 participant
- Score 49: 2 participants
- Score 47: 4 participants
- Score 32: 8 participants
- Score 11: 4 participants
- Score 0: 2 participants
Subtask 1

• Use array to store the elements of A and B
  For every query
    Set ans to INF
    For every element \((S_A, T_A)\) of A
      For every element \((S_B, T_B)\) of B
        Update ans if \(S_A \times S_B + T_A + T_B < ans\)

• Time complexity: \(O(N^3)\)
• Expected score: 11
Subtask 2

- When we look at the solution for subtask 1, we can see that many of the pairs are calculated repeatedly which is not necessary.
- Instead, we can maintain the current answer.
- When a new plan is added, we try to see if this new plan gives a smaller answer and update current answer.
- This can reduce the overall time complexity to $O(N^2)$
Subtask 2

For every query
  if it is type 1
    if the plan is for part A
      insert the plan into A
    for every plan in B
      try to update answer
    if the plan is for part B
      insert the plan into B
    for every plan in A
      try to update answer
  if it is type 2
    output answer
Subtask 2

• Some contestant even manage to pass subtask 3 using this algorithm.
Observations

• Consider the following simplified version.
• Given a set A.
• Given a point \((S_B, T_B)\) in B.
• You are to find \(\min(S_A \times S_B + T_A + T_B)\) where \((S_A, T_A) \in A\)

• It is obvious that if \(S_A = S'_A\) and \(T_A < T'_A\) then choosing \((S_A, T_A)\) must be better than choosing \((S'_A, T'_A)\), so we can remove \((S'_A, T'_A)\), let’s assume \(S_A\) are distinct from now on.
Observations

• Recall what we have learnt in senior secondary school.
• To find $\min(x \times S_B + y + T_B)$, we can use linear programming.
• Let’s draw a line with slop $-S_B$
slope = -1

(1, 4)  (3, 2)  (4, 5)  (6, 3)  (7, 1)
Observations

• We know that the point which touches the line with smallest y-intercept is corresponding to the minimum solution.

• So we know that when $S_B = 1$
  • $S_B \times 1 + T_B + 4$ is the minimum

• But, can this really speed up our solution?
Observations

• By drawing lines with different slope, we found that some points will never optimal.
Observations

- Let $H$ be the set useful points
- (For those who knows convex hull: the set $H$ is actually a lower hull)

- We can see that if $S_B$ is very big -> the line is very steep -> we will always choose the leftmost point.
- So the leftmost point must be useful.
Observations

• Let $r = (r_x, r_y)$ be the right most point in $H$.

• We can see that a point on the right side of $r$ which makes the smallest slope with $r$ will also be useful.

• So how can we use this fact to build the set $H$?
Observations

• Consider joining the useful points one by one, we can see that the slope is increasing.
• So we may use a stack to build H.
• Let p1 be the leftmost point.
• Initially, we will p1 and the point p2 which makes a smallest slope with p1 (we can find this point just using a linear search)
• Then we know that any point with x-coordinates between p1 and p2 must not be a useful point, so we can also remove them.
Observations

• Then we can use a stack to build H.

Push p1 and p2 into H
r = p2, r2 = p1
For any remaining point p in A from left to right up to (A, 0)
  while slope(r2, r) > slope(r, p)
    pop H
    r = last of H
    r2 = second last of H
  push p into H
Observations

• After building up H, we can find an optimal useful point for any given slope very fast.
• How?
Observations

• We can see that if the slope is very small (steeper), then we will choose the left most point.

• If the slope is bigger than the slope between two consecutive points in H, we can see that choosing the left point must not be an optimal solution.

• So we can perform a binary search on H to find the optimal point.

• Since we can reuse H for every point in B, so we can find the optimal solution between all pairs of plans in $O(N \log N)$
Subtask 2*

For every query,

use the above method to build $H_A$ and $H_B$.

For every point $p_A$ in $A$  
do a binary search on $H_B$ and update answer

For every point $p_B$ in $B$  
do a binary search on $H_A$ and update answer

- Time complexity: $O(N^2 \log N)$
- Expected score: 32
Subtask 2*

• Actually, you can replace the binary search by linear search if you have sorted the points.

• A small trick here,

• You can include the plan (INF, 0) to A and B

• This can make sure that there are always at least 2 points in A and B so that we can avoid the case that our H will have only one point
Subtask 3

• Does the constraint looks strange?

• As you may guess by looking at the constraint, we can do square root decomposition on this question.
Subtask 3

• We will maintain the current answer and a partially online version of $H$.

• We will rebuild $H_A$ and $H_B$ in every $\sqrt{N}$ queries

• For every type 1 query of inserting points to $A$ (or $B$),
  • We can insert the point to $\text{new}_A$ (or $\text{new}_B$)
  • Do a binary search on $H_B$ and update answer
  • Do a linear search on $\text{new}_B$ and update answer

• For every type 2 query, we can simply output current answer.
Subtask 3

• Rebuild part
  • Single time : \( O(N \log N) \)
  • We will perform \( O(\sqrt{N}) \) times.
  • Overall : \( O(N\sqrt{N} \log N) \)

• Query & update part
  • Single time : \( O(\log N + \sqrt{N}) = O(\sqrt{N}) \)
  • We will perform \( O(N) \) times.
  • Overall : \( O(N\sqrt{N}) \)

• Time complexity: \( O(N\sqrt{N} \log N) \)

• Expected score: 47
Subtask 4

• The constraint is just a bit larger than that of subtask 3, maybe getting rid of a log N in the solution is useful.

• We can see that the log N only appears in the rebuild part and that is because of sorting.

• Can we maintain the order of data using less time?
Subtask 4

• Can we maintain the order of data using less time?
  • Yes, we can!

• When we insert points into new, we can insert according to order, which is $O(N^{0.5})$

• Then in the rebuild part, we do not need to sort the points but merge two arrays of sorted points into one.
Subtask 4

• Rebuild part
  • Single time : O(N)
  • We will perform O(√N) times.
  • Overall : O(N√N)

• Query & update part
  • Single time : O(\log N + √N) = O(√N)
  • We will perform O(N) times.
  • Overall : O(N√N)

• Time complexity: O(N√N)
• Expected score: 59
Observations

• We can see that adding new points will only make some useful point useless but not useless point useful
Observations

• Suppose $R$ is the maximum coordinate of the points.

• There are at most $O(\sqrt{R})$ points in $H$.

• Consider the points $(1, R), (2, R-1), (3, R-1-2)...$

• Slope $-1$ to $\frac{\sqrt{8R+1}-1}{2}$ is included in $H$
  • so there are at most $O(\sqrt{R})$ points in $H$. 
Subtask 5

• Base on a solution of subtask 2, 3 or 4.
• When a point is removed in building H, we remove it totally so that we will never consider this point when building H later.

• The time complexity will be the same but the runtime will be around 10 times faster (for R <= 10000)
Subtask 6

• There are 2 solutions for subtask 6.
• The first one is to maintain the set $H$ online.
• We can achieve this by storing $H$ in a set.
• When we try to insert a new point $(s, t)$ into the set, we may need to remove a consecutive part in $H$.
• This involves usage of iterator for set and edge cases handling, seems difficult to discuss here...
• So try to implement yourself, this can train your coding skills.
Subtask 6

• Suppose we can maintain H online with $O(\log N)$ time.
• We should also maintain the current best answer like in subtask 3.
• We just replace the query & update part by using the set H.

• Ps. Don’t forget that we have to maintain plan from two parts, ie. 2Hs

• Time complexity: $O(N \log N)$
• Expected score: 100 if no bugs
• Solution: https://judge.hkoi.org/submission/284650/details?sharing=wCgqOGiNL6HtU1e3UhX1o4ZWE
Subtask 6

• The other one is easier
• Recall what you have learnt yesterday again....
Subtask 6

• CDQ D&C seems useful here.
• But how?
Subtask 6

• Actually, type 2 is NOT meaningful in this algorithm.
• We should treat every insertion of point as both update and query.
• So we will store the answer on type 1 query.
• To find the answer of a type 2 query, we just need to find the running min.

• Time complexity: $O(N \log^2 N)$
• Expected score: 100
• Solution: https://judge.hkoi.org/submission/284647/details?sharing=iDXniTG2axPNrIburijL08ZitU
Subtask 6

• We can even get rid of one log by using linear search and merge sort.

• Time complexity: $O(N \log N)$

• Expected score: 100

• Solution: https://judge.hkoi.org/submission/284632/details?sharing=bKqdUhF UWKvbsCB6pl1Z0qw5G0

• However, the runtime is not improved.
Civilizations

Tony Wong
Hong Kong Olympiad in Informatics
2018-07-02

Source: KOI 2017 (High School)
Task Description

• Initially there are K painted cells in a N x N grid

• Each year the painted cells expand to their adjacent cells (4 directions)

• Find number of years until the painted cells become connected
Subtask 1: $O(N^3)$

- Repeat (at most N times)
  - Longest path is corner to the opposite corner

- Check if the painted cells are connected using DFS or BFS $O(N^2)$

- For each painted cell, expand to adjacent cells $O(N^2)$
Subtask 2: $O(N^2 \lg N)$

• Note that in Subtask 1 we try ans = 1, 2, 3, ...

• Instead, we can use binary search on answer

• After fixing the time $t$, we can use BFS from each civilization’s origin, and mark all nodes with distance $\leq t$

• Finally, check if the civilizations are connected
Subtask 3: $O(N^2)$

- To solve the problem, we optimize subtask 1’s solution
- Repeat (at most $N$ times)
  - Check if the newly painted cells are connected using DFS or BFS disjoint-set union-find $O(N)$
  - For each newly painted cell, expand to adjacent cells $O(N)$
Subtask 3: \(O(N^2)\)

- Use a vector to store “newly painted cells”
- For each input civilization:
  - Put the origin into the vector
  - Create a disjoint set with \(\text{group}_i = 1\)
- For year = 0, 1, 2 ....
  - For each last year’s newly painted cells, union them with the adjacent cells
    - When merging disjoint set \(a\) into \(b\): \(\text{parent}[b] = a\)
      - “transfer” the civilization count: \(\text{group}_a += \text{group}_b\)
  - If a disjoint set contains \(K\) civilizations, output day
  - For every last day’s newly painted cells, add their unvisited adjacent cells into another vector
Year 0

Merge

Expand

<table>
<thead>
<tr>
<th>Disjoint Set</th>
<th>Size</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Year 1

Merge

Expand

<table>
<thead>
<tr>
<th>Disjoint Set</th>
<th>Size</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>--</td>
<td>3</td>
</tr>
</tbody>
</table>
Year 2

Merge

Expand

<table>
<thead>
<tr>
<th>Disjoint Set</th>
<th>Size</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>--</td>
<td>3</td>
</tr>
</tbody>
</table>
Year 4

Merge

Expand

<table>
<thead>
<tr>
<th>Disjoint Set</th>
<th>Size</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>--</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>--</td>
<td>3</td>
</tr>
</tbody>
</table>
Score distribution

Note: Area weighted by score