ENS Lyon Training Camp Day 04. Problem Analysis

Maxim Buzdalov

ITMO UNIVERSITY

October 30, 2015

Problem A. Number of paths in acyclic graph

Statement

- Given an acyclic graph with N vertices
- Find the number of paths from 1 to N

Problem A. Number of paths in acyclic graph

Statement

- Given an acyclic graph with N vertices
- Find the number of paths from 1 to N

Solution

- $A(i)$ — the number of paths from 1 to i
- $A(1) = 1$, $A(i \neq 1) =$ sum of all $A(j)$ such that $j \to i$ is an edge
- Time and space complexity: $O(N)$

Problem B. Knapsack

Statement

- N items, each has weight w_i and cost c_i
- A knapsack with max weight of W
- Find the subset of items which fit the knapsack and have maximum cost

Problem B. Knapsack

Solution

- $C(i,j)$ — the maximum cost you can have by using some items from $[1;i]$ with total weight exactly j
- $B(i,j)$ — whether you should use the item i
- Boundary: $C(x,0) = 0$ for all x
- $C(i,j)$ is a maximum of:
  - $C(i-1,j) \leftarrow$ don't get the item i
  - $C(i-1,j-w_i) + c_i \leftarrow$ get the item i
- Answer: the max of $C(n,j)$ for $j \in [1;W]$
- Time and space complexity: $O(NW)$

Problem C. Longest common subsequence

Statement

- Given two sequences $A_{[1;N]}$ and $B_{[1;M]}$
- What is their longest common subsequence?

Problem C. Longest common subsequence

Solution

- $L(i,j)$ — the LCS length for $A_{[1;i]}$ and $B_{[1;j]}$
- $L(i,j) = \max(L(i-1,j), L(i,j-1))$
- If $A_i = B_j$, then
  $L(i,j) \leftarrow \max(L(i,j), 1 + L(i-1,j-1))$

Problem C. Longest common subsequence

Solution

- $L(i,j)$ — the LCS length for $A_{[1;i]}$ and $B_{[1;j]}$
- $L(i,j) = \max(L(i-1,j), L(i,j-1))$
- If $A_i = B_j$, then $L(i,j) \leftarrow \max(L(i,j), 1 + L(i-1,j-1))$
- Restore an answer: $B(i,j) = \{i^-, j^-, ij^-\}$ (which way to move)
- Can be done without that

Problem C. Longest common subsequence

Solution

- $L(i,j)$ — the LCS length for $A_{[1;i]}$ and $B_{[1;j]}$
- $L(i,j) = \max(L(i-1,j), L(i,j-1))$
- If $A_i = B_j$, then $L(i,j) \leftarrow \max(L(i,j), 1 + L(i-1,j-1))$
- Restore an answer: $B(i,j) = \{i^-, j^-, ij^-\}$ (which way to move)
- Can be done without that
- Time and space complexity: $O(NM)$

Problem D. Levenshtein distance

Statement

- Given two strings A and B
- Operations: add symbol, remove symbol, replace symbol
- What is the shortest sequence of operations which transforms A to B?

Problem D. Levenshtein distance

Solution

- $D(i,j)$ — the Levenshtein distance between $A_{[1;i]}$ and $B_{[1;j]}$
- Initialization: $D(i,0) = i$, $D(0,j) = j$
- $D(i,j)$: the general case:
  - A_i can be removed: $D(i,j) \leftarrow 1 + D(i-1,j)$
  - B_j can be added: $D(i,j) \leftarrow 1 + D(i,j-1)$
  - A_i can be replaced by B_j: $D(i,j) \leftarrow 1 + D(i-1,j-1)$
  - If $A_i = B_j$, $D(i,j) \leftarrow D(i-1,j-1)$
- Time and space complexity: $O(|A||B|)$

Problem E. Longest increasing subsequence

Statement

- Given a sequence of integers $A_{[1;N]}$
- Find a longest increasing subsequence

Problem E. Longest increasing subsequence

Solution

- $D(i)$ — the length of a LIS which contains i-th element
- $B(i)$ — a pointer to the previous element
- How to compute?
  - check all $j < i$ where $A_j < A_i$
  - if $D(j) + 1 > D(i)$, update $D(i)$ and $B(i)$
- Time complexity: $O(N^2)$
- Space complexity: $O(N)$

Problem F. Maximal weight matching in tree

Statement

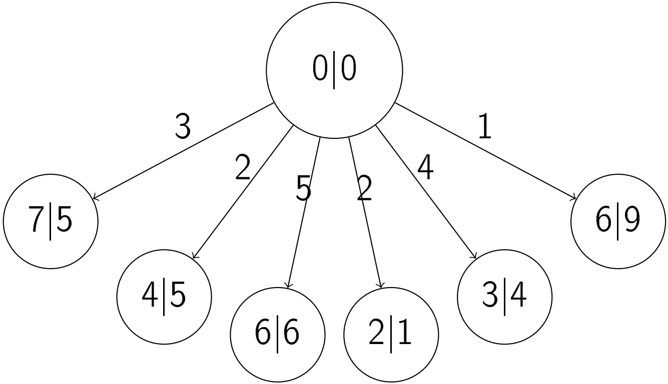
- Given a tree with weights on edges
- Find a maximum weight matching

Solution

- Turn an arbitrary vertex into a root
- Dynamic programming: maximum weight matching for a subtree
- Two cases: subtree root is or is not paired with a child

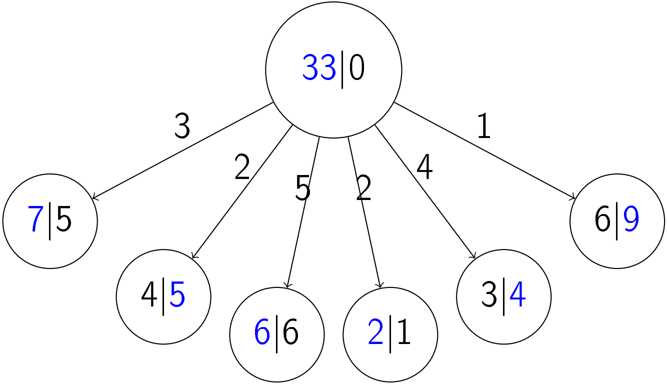
Problem F. Solution

- First, assume that the root is not paired
- Sum best values from children



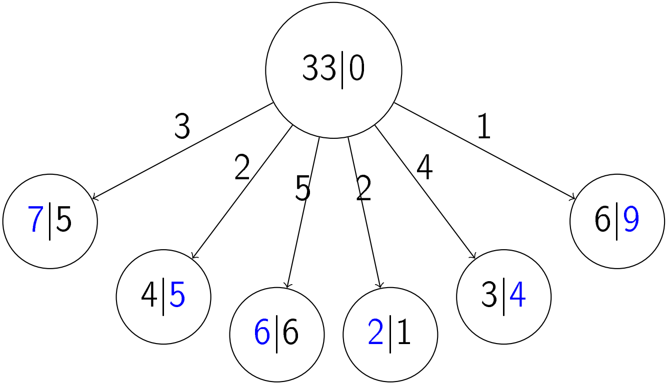
Problem F. Solution

- First, assume that the root is not paired
- Sum best values from children



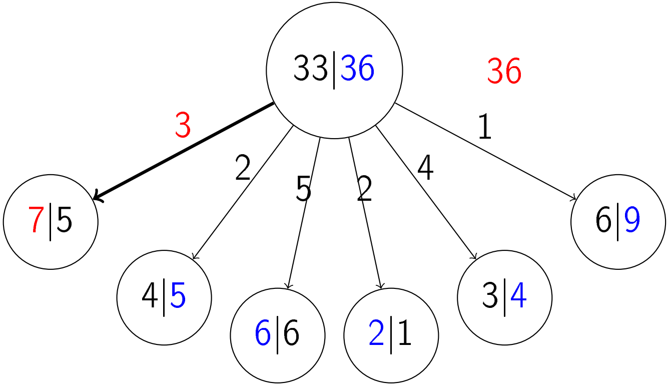
Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



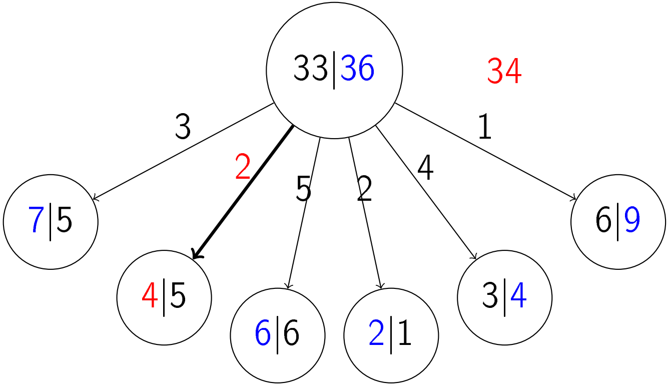
Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



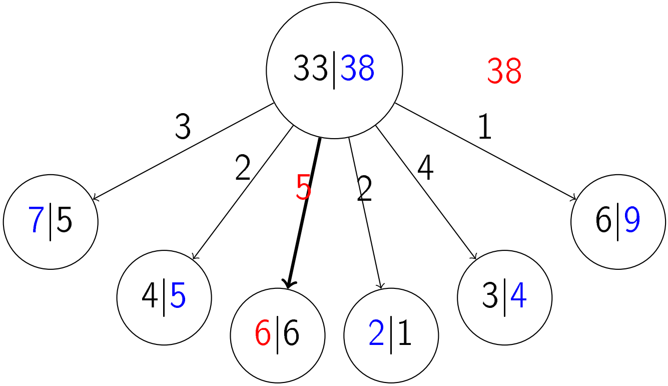
Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



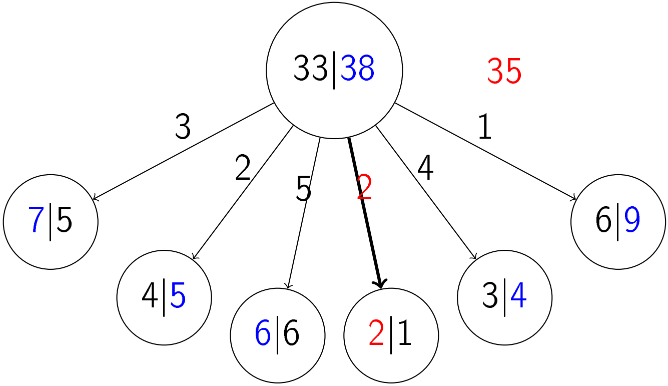
Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



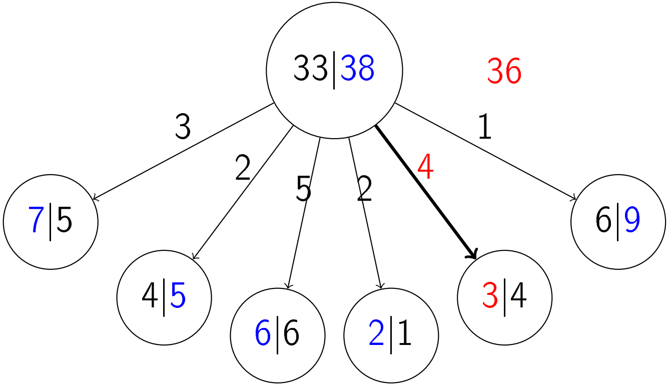
Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



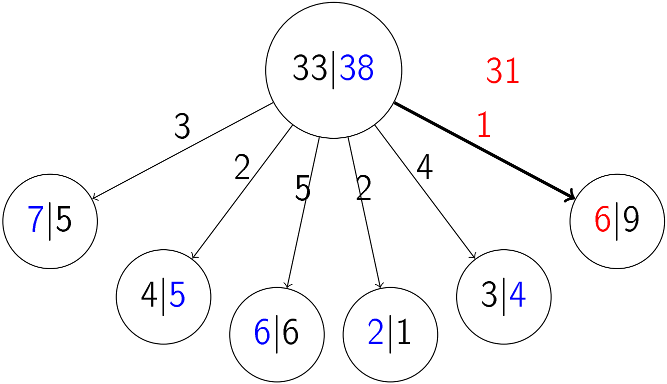
Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



Problem F. Solution

- Second, pair root with every child in turn
- For the paired child use unpaired DP value



Problem G. Matrix multiplication

Statement

- Given N matrices of size $A_1 \times A_2$, $A_2 \times A_3$, $\ldots$, $A_N \times A_{N+1}$
- Assume that it takes $X \cdot Y \cdot Z$ operations to multiply a matrix $X \times Y$ by a matrix $Y \times Z$
- Find the positioning of parentheses such that the total number of operations is minimal

Problem G. Matrix multiplication

Solution

- $D(i, j)$ — the minimum number of operations to multiply matrices from i-th to j-th
- $D(i, i) = 0$
- $D(i, j) = \min_{i \le k < j}$ $(D(i, k) \cdot D(k + 1, j) + A_i \cdot A_{k+1} \cdot A_{j+1})$
- Restoring the answer: $B(i, j)$ is the optimum k from above
- Time complexity: $O(N^3)$
- Space complexity: $O(N^2)$

Problem H. Longest subpalindrome

Statement

- Given a string S, find its longest subsequence which is a palindrome

Problem H. Longest subpalindrome

Statement

- Given a string S, find its longest subsequence which is a palindrome

Solution

- $D(i,j)$ — the answer for $S_{[i;j]}$
- $D(i,i) = 1$
- Recomputation:
  - $D(i,j) \leftarrow \max(D(i+1,j), D(i,j-1))$
  - If $S_i = S_j$, then $D(i,j) \leftarrow 2 + D(i+1,j-1)$
- Time and space complexity: $O(|S|^2)$

Problem I. Traveling salesman problem

Statement

- N cities, M roads
- d_{ij} — the length of a road between cities i and j (may be ∞)
- Find a shortest path which visits every city exactly once

Problem I. Solution

- $D(S, i)$ — the shortest path from vertex 1 to vertex i which visits every city from a set S exactly once
- $D(S, i) = \min_{j \in S \setminus \{i\}} D(S \setminus \{i\}, j) + d_{ij}$

Problem I. Solution

- $D(S, i)$ — the shortest path from vertex 1 to vertex i which visits every city from a set S exactly once
- $D(S, i) = \min_{j \in S \setminus \{i\}} D(S \setminus \{i\}, j) + d_{ij}$
- To find the answer:
  - test all endpoints i and j
  - test all vertex sets S which include 1 and i
  - update the answer with $D(S, i) + D(\{1\} \cup \overline{S}, j)$

Problem I. Solution

- $D(S, i)$ — the shortest path from vertex 1 to vertex i which visits every city from a set S exactly once
- $D(S, i) = \min_{j \in S \setminus \{i\}} D(S \setminus \{i\}, j) + d_{ij}$
- To find the answer:
  - test all endpoints i and j
  - test all vertex sets S which include 1 and i
  - update the answer with $D(S, i) + D(\{1\} \cup \overline{S}, j)$
- Implementation detail: use integers for vertex sets (bit masks)
- Time complexity: $O(2^N \cdot N^2)$

Problem J. Bracket subsequences

Statement

- Given a bracket sequence
- How many different subsequences are regular bracket sequences?

Insights

- First, big integers:
  the answer can have 60 digits
- Different subsequences:
  need count each one exactly once

Problem J. Solution (1/2)

- A non-ambiguous context-free grammar for regular bracket sequences:
  - $S \leftarrow \varepsilon \mid (S)S$
  - either an empty sequence, or the first opening bracket, its closing bracket and the remaining parts
- This enables a dynamic programming idea: counting the answer for sequence segments

Problem J. Solution (2/2)

Dynamic programming

- $A(i, j)$: answer for sequence segment $S_{[i;j]}$
- $A(i+1, i) = 1$: an empty sequence
- $S_i =$ ')': $A(i, j) = A(i+1, j)$
- Otherwise:
  - check all closing bracket indices $k \in [i+1; j]$
  - $A(i, j) \leftarrow A(i, j) + A(i+1, k-1) \cdot A(k+1, j)$

Problem J. Solution (2/2)

Dynamic programming

- $A(i,j)$: answer for sequence segment $S_{[i;j]}$
- $A(i+1,i) = 1$: an empty sequence
- $S_i =$ ’)’: $A(i,j) = A(i+1,j)$
- Otherwise:
  - check all closing bracket indices $k \in [i+1;j]$
  - $A(i,j) \leftarrow A(i,j) + A(i+1,k-1) \cdot A(k+1,j)$
- Not really, as you may count some subsequences twice (example: ’(())()’)
  - ())()(()()())
  - ())()((()))())

Problem J. Solution (2/2)

Dynamic programming

- $A(i, j)$: answer for sequence segment $S_{[i;j]}$
- $A(i+1, i) = 1$: an empty sequence
- $S_i =$ ')': $A(i, j) = A(i+1, j)$
- Otherwise:
  - check all closing bracket indices $k \in [i+1; j]$
  - $A(i, j) \leftarrow A(i, j) + (A(i+1, k-1) - A(i+1, k'-1)) \cdot A(k+1, j)$ where k' is the previous closing bracket index

Problem K. String decomposition

Statement

- Represent the given string $S = S_1^{d_1} \ldots S_k^{d_k}$, where $A^b = AA \ldots A$ b times, such that $\sum_i d_i$ is minimum possible

Solution

- $D_1(i, j)$ — the maximum d in $S_{[i;j]} = T^d$
  - using prefix function or z-function for each i separately, running time: $O(|S|^2)$
- $D_2(i)$ — the answer for $S_{[1;i]}$
  - minimum of $D_2(j) + D_1(j + 1, i)$ for all $0 \le j < i$