TDDD95 - Seminar 1

February 1, 2019

1 problems

1.1 Spiderman's workout

Dynamic programming.

for *Optimal substructure* and *overlapping subproblems*

Optimal substructure means there is a solution that can be obtained by combining optimal solutions to it's subproblem.

Overlapping subproblems means that finding a solution involves solving the same subproblem several times.

1.2 Ljutna

Solved.

1.3 Help!

Iterate over pairs, if there are a word a placeholder, replace all placeholders of the same type. Then reiterate and see if there are any unmatching words.

1.4 Aspen avenue

Sort according to left side of the road.

Define a *decision state* as having assigned x trees to the left side, and y trees to the right side. Now, given that we want to assign the tree with index $i = x+y+1$, do exactly what we did with the spiderman problem: Try the two options that we have: Left of right(brute-force)

Since a subproblem can be defined by it's decision state, we can use a table to stroe solutions to subproblems using an array dp[1000][1000]

Time complexity: $O(N^2)$

2 Time Limits and Computational Complexity

3 Basic data structures

3.1 Linear data structures

- Pair, Tuple (C++11)
- Static array
- Vector (ArrayList or Vector)
- bitset (BitSet)
- stack (Stack)
- queue
- dequeue

3.2 Rip more stuff

4 Lab problems

4.1 Interval Cover

Problem: Given an interval $[L, R]$ and a set of other intervals $[l_1, r_1], \ldots, [l_n, r_n]$, find the minimal number of such intervals that are needed to cover $[L, R]$.

Super simple greedy algorithm:

- "do as long as you can": Cover the "left part" L of the target interval $[L, R]$ with "the interval $[l, r]$ that covers L and has the largest r of all such intervals" (then set $[L, R] := [r, R]$)

4.2 Knapsack

Solved by dynamic programs. "Should I put item i in the knapsack if I carry w weight"?

State: $[i][w]$

Generalizes **MANY** well-known optimiation problems.

4.3 Disjoint set

The *disjoint set* is a data structure for storing a set of disjoing sets where it is very effiecient $\approx O(1)$ to *find* which set an element belongs to and to merge ("unify") two sets.

The disjoin sets are represented by a *forest of trees*, where the root of a tree is the representative element for that set.

To improve the performance use path compression.

Example usage: Finding connected components in a unidercted graph or Kruskal's algorithm for finding a Minimum Spanning Tree.

4.4 Fenwick Tree

A Fenwick tree is an efficient data structure for computing range sum queries with updates, both in $O(\log n)$

Naively, $O(n)$, so this is a great improvement

A Fenwick tree only stores range sums, not the original values.

Basic idea: Each intiger can be represented as sum of powers of two. In the same way, cumulative frequency can be represented as a sum of sets of subfrequencies. In our case, each set contains

Tips

- Read original paper by Fenwick
- Implementing only takes a few lines of code. Snipe online code
- More important how to use than how to implement
- Also known as BIT (Binary ... tree)

5 Tips

Debugging: If you get stuck for too long, ask for test dtaa that breaks algorithm.

If you're still stuck, take a break with a new problem.

Include everything! include ¡bits/stdc++.hn¿