CSC373: Algorithm Design, Analysis and Complexity
Winter/Spring 2020

Allan Borodin and Sara Rahmati

Week of January 27-31, 2020

Announcements

- We have added one final question to Assignment 1. It is (we believe) an easy dynamic programming problem.
- We are posting Sara Rahmati's lecture slides. However, as Sara is using many slides from Kevin Wayne's web site, for copyright reasons we are password protecting Sara's slides. We will repeat the password in lecture.
- The good news is that many students submitted solutions to the bonus question. We understand that students who submitted after the first submission may have had a class or were in transit and hence could not be first to respond. So we will give the bonus to everyone who submitted a correct solution. Everyone (whether you submitted a solution or not) should be able to find an example where EFT coloring is not optimal.

  Question: What is the largest EFT/OPT ratio you can obtain?

Important announcement

Cheating in quiz 1 and our response: We have learned that some students in the 4-5 section were passing on the content of the quiz to students taking the quiz in the 5-6 section. We will be taking steps to avoid this which will mean some inconvenience to everyone. Although we may start giving different quizzes to the two sections, we will also adopt procedures similar to the final exams,

The quiz for the 5-6 sections will begin promptly at 5:10. Students arriving late will not be allowed to take the quiz. Students in the 4-5 sections will take the quiz at the end of the hour and cannot leave until the end of the hour after quizzes have been collected. No cell phones will be allowed during the quiz. We may take additional measures.

We take acadmic offenses seriously. You could wind up with a serious academic offense on your trtanscript or worse.

Students who act properly should not feel they are at a disadvantage because others are cheating. Because the quiz has been compromised, we will give everyone the 2.5% so that no one is disadvantaged.

This weeks agenda

- Finish the slides from the last lecture on edit distance.
- The travelling salesman (salesperson) problem (TSP)
- We will use Sara's DP slides to discuss:
  - The RNA secondary structure problem.
  - Detecting (and finding) a negative weight cycle in a directed graph
- On the board we will discuss the weighted independent set problem for trees. (This is in Chapter 10 of the KT text.) This leads to an optimal algorithm for the weighted maximum independent set problem for a class of graphs called "bounded tree-width".
- Some concluding remarks on dynamic programming.

Analyzing the complexity of a DP algorithm

Although we tend to design DP algorithms recursively, it is not a good idea to try to analyze the complexity of DP algorithms by analyzing the corrreponding recurrence (as we did in the analysis of divide and conquer algorithms). The complication for DP algorithms is that if we try to analyze the recurrence, the recurrence ignores the memoization that needs to be done if the algorithm is being executed as a recursive algorithm.

Here is a simple way to obtain an upper bound on the complexity of a DP algorithm. Usually we design a DP algorithm by creating an arrray where each entry of the array can be efficiently computed given that "previous" entries of the array arev already computed. So an upper bound on the complexity is "(the size of the array) times the (cost for each entry given previous entrfues are already computed)".

The sequence alignment (edit distance) problem

The edit distance problem

- Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ over some finite alphabet S.
- **Goal:** find the best way to "match" these two strings.

- Variants of this problem occur often in bio-informatics as well as in spell checking.
- Sometimes this is cast as a maximization problem.
- We will view it as a minimization problem by defining different distance measures and matching symbols so as to minimize this distance.

A simple distance measure

- Suppose we can delete symbols and match symbols.
- We can have a cost $d(a)$ to delete a symbol a in S, and a cost $m(a, b)$ to match symbol a with symbol b (where we would normally assume $m(a, a) = 0$).
- As in any DP we consider an optimal solution and let's consider whether or not we will match the rightmost symbols of X and Y or delete a symbol.

The DP arrays

- The semantic array:

$$E[i,j] = \text{the cost of an optimal match of } x_1 \dots x_i \text{ and } y_1 \dots y_j.$$

- The computational array:

$$E'[i,j] = \begin{cases} 0 & \text{if } i = j = 0 \\ d(y_j) + E'[i, j-1] & \text{if } i = 0 \text{ and } j > 0 \\ d(x_i) + E'[i-1, j] & \text{if } i > 0 \text{ and } j = 0 \\ \min\{A, B, C\} & \text{otherwise} \end{cases}$$

where $A = m(x_i, y_j) + E'[i-1, j-1]$, $B = d(x_i) + E'[i-1, j]$, and $C = d(y_j) + E'[i, j-1]$.

- As a simple variation of edit distance we consider the maximization problem where each "match" of "compatible" a and b has profit 1 (resp. $v(a, b)$) and all deletions and mismatches have 0 profit.
- This is a special case of unweighted (resp. weighted) bipartite graph matching where edges cannot cross.

The traveling salesman problem (TSP)

We recall from last week that computing the maximum cost (i.e. profit) of a simple path is a generalization of the Hamiltonian path problem which is a variant of the traveling salesman problem.

Traveling salesman problem (TSP)

Given a graph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}_{\geq 0}$ determine if the cost of a simple cycle containing all the nodes (i.e. cycle length is $n = |V|$) assuming the graph has such a Hamiltonian cycle.

Without loss of generality, we can assume a complete graph (using $c(e) = \infty$ for any missing edges).

It is is roughly equivalent to consider the least cost Hamiltonian path problem. Namely, finding a least cost *simple* path of length *exactly* (and NOT at most) length $n - 1$ from some given starting node u. For the same reason as in the maximum cost path discussion, the least cost Hamiltonian path problem cannot be obtained by modifying the least cost path DP. Namely, we cannot dismiss the possibility of cycles in the path.

Not all exponentials are equal; using DP to obtain a better exponential time algorithm

A naive way to compute the least cost Hamiltonian path problem (with some given initial node u) is to consider all $(n-1)!$ simple paths of length $n-1$. As we noted last week, $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$. We also mentioned that by using an appropriate DP, we can reduce that time complexity to $O(n^2 2^n)$ which is still of course exponential but grows much slower than the factorial function $(n!)$.

Here is the idea as expressed in the following semantic array: For each subset $S \subseteq V$ with $u \in S, v \notin S$
$C[S, v]$ is the least cost simple path from u to v containing each node in S exactly once.

Computing the entries of $C[S, v]$

I am now going to just use C and not C' since by now I hope the distinction between "what we want to compute" and "how we are going to compute it" is hopefully clear. We compute $C[S, v]$ "inductively as follows:

If $|S| = 1$, then $C[S, v] = c(u, v)$ % S must be $\{u\}$
Else if $|S| > 1$, then $C[S, v] = \min_{x \notin S} C[S, x] + c(x, v)$

We note that the least cost Hamiltonian path problem is "NP-hard to approximate" to any constant whereas there are efficient approximations if the cost function satisfies the triangle inequality.

DP concluding remarks

- In DP algorithms one usually has to first generalize the problem (as we did more or less to some extent for all problems considered). Sometimes this generalization is not at all obvious.

- What is the difference between divide and conquer and DP?

  - In divide and conquer the recursion tree never encounters a subproblem more than once.
  - In DP, we need memoization (or an iterative implementation) as a given subproblem can be encountered many times leading to exponential time complexity if done without memoization.
  - See also the comment on page 169 of DPV as to why in some cases memoization pays off since we do not necessaily have to compute every possible subproblem. (Recall also the comment by Dai Tri Man Le.)

Dynamic Programming Over Intervals

Notation. $OPT(i, j)$ = maximum number of base pairs in a secondary structure of the substring $b_i b_{i+1} \ldots b_j$.

- Case 1. If $i \geq j - 4$.
  - $OPT(i, j) = 0$ by no-sharp turns condition.

- Case 2. Base b_j is not involved in a pair.
  - $OPT(i, j) = OPT(i, j-1)$

- Case 3. Base b_j pairs with b_t for some $i \leq t < j - 4$.
  - non-crossing constraint decouples resulting sub-problems
  - $OPT(i, j) = 1 + \max_t \{ OPT(i, t-1) + OPT(t+1, j-1) \}$

take max over t such that $i \leq t < j-4$ and b_t and b_j are Watson-Crick complements

Remark. Same core idea in CKY algorithm to parse context-free grammars.