Tutorial 3

Dynamic programming

Problem 15.3 (405): Give an $O(n^2)$-time algorithm for finding an optimal bitonic traveling-salesman tour. Scan left to right, maintaining optimal possibilities for the two parts of the tour.

Solution: Sort the points by x-coordinate, left to right, in $O(n \log n)$ time. Let the sorted points be $p_1, p_2, \ldots, p_n$.

Subproblems: bitonic paths P_{ij}, where $i \leq j$, that includes points $p_1, \ldots, p_j$. It starts at p_i, goes strictly left to p_1, and then goes strictly right to p_j. Going strictly left means that each point in the path has a lower x-coordinate than the previous point (the indices of the sorted points form a strictly decreasing sequence). Likewise, going strictly right means that the indices of the sorted points form a strictly increasing sequence. Note that p_j is the rightmost point in P_{ij} and is on the rightgoing subpath. The leftgoing subpath may be degenerate, consisting of just p_1.

Let $|p_i p_j|$ be the euclidean distance between p_i and p_j, and $b[i,j]$, for $1 \leq i \leq j \leq n$, be the length of the shortest bitonic path P_{ij}. Since the leftgoing subpath may be degenerate, we can easily compute all values $b[1,j]$. The only value of $b[i,i]$ that we will need is $b[n,n]$, which is the length of the shortest bitonic tour. Hence:

$$\begin{aligned} b[1,2] &= |p_1 p_2| \\ b[i,j] &= b[i,j-1] + |p_{j-1} p_j|, \text{ for } i < j-1 \\ b[j-1,j] &= \min_{1 \leq k < j-1} \{b[k,j-1] + |p_k p_j|\} \end{aligned}$$

Any bitonic path ending at p_2 has p_2 as its rightmost point, so it consists only of p_1 and p_2. Its length is therefore $|p_1 p_2|$.

Consider a shortest bitonic path P_{ij}. If p_{j-1} is on its rightgoing subpath, then it immediately preceeds p_j. The subpath from p_1 to p_{j-1} must be a shortest subpath $P_{i,j-1}$, since we otherwise could replace it to get a shorter bitonic path than P_{ij}. The length of P_{ij} is therefore given by $b[i,j-1] + |p_{j-1} p_j|$.

If p_{j-1} is on the leftgoing subpath, then it must be its rightmost point, so $i = j-1$. Then p_j has an immediate predecessor p_k, for $k < j-1$, on the rightgoing subpath. Optimal substructure again applies: the subpath from p_k to p_{j-1} must be a shortest bitonic path $P_{k,j-1}$. The length of P_{ij} is therefore given by $\min_{1 \leq k < j-1} \{b[k,j-1] + |p_k p_j|\}$.

In an optimal bitonic tour, one of the points adjacent to p_n must be p_{n-1}, so $b[n,n] = b[n-1,n] + |p_{n-1} p_n|$. To reconstruct the points on the shortest bitonic tour, we define $r[i,j]$ to be the index of the immediate predecessor of p_j on the shortest bitonic path P_{ij}. Because the immediate predecessor of p_2 on $P_{1,2}$ is p_1 , we know that $r[1,2]$ must be 1.

Since we iterate over i and j the time is $O(n^2)$, which dominates the initial sorting time.

Problem 15-6 (408): In a company with a hierarchical structure each employee has a conviviality rating. Give an efficient algorithm to make up the guest list to a company party, maximizing the sum of the conviviality ratings without inviting both an empoyee and the immediate supervisor.

Solution: Each employee has a conviviality rating r. Use dynamic programming to compute the largest sum R of ratings such that persons on adjacent levels in the hierarchy are not invited. Start the computation at the leaves. Each employee is a node, v, and a root in a subtree $T(v)$. Its children $c[v]$ are the

nodes directly below, and its grandchildren $gc[v]$ are those one level further down. If v is a leaf then $R(v) = r[v]$. Otherwise the maximum rating $R(v)$ of $T(v)$ is given by

$$R(v) = \max\{\,(r[v] + \sum_{u \in gc[v]} R(u)),\ \sum_{u \in c[v]} R(u)\}$$

Either v is included in the guest list for $T(v)$ (first term) or not. If there are no grandchildren, let the corresponding terms have rating 0. The value at the root gives the answer. By updating lists of pointers to the children or grandchildren that gave $R(v)$, we can at the end identify the party guests by traversing these lists from the root. The rating of a node is looked up twice, as child and grandchild, when computing the R-value of another node. And an R-value is never changed. Hence the total time is $O(n)$, where n is the number of employee.

Problem 25.2-8 (700): Give an $O(VE)$ algorithm to compute the transitive closure of a directed graph.

Solution: Run an $O(E)$-time graph search (like DFS) from each vertex. This determines all the vertices v_j that can be reached from each origin v_i (i.e. sets the bit t_{ij} to 1). Since there are $|V|$ vertices to start from we get $O(VE)$ time.

Note: The transitive closure can be computed in $O(M(n))$ time, where $M(n)$ is the time to multiply two $n \times n$ matrices. I will present the algorithm if anyone is interested.

Amortized analysis

Problem 17.1-3 (456): Perform n operations on a data structure, where the ith operation costs i if i is a power of 2, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

Solution: The cost of the $i$the operation

$$c_i = \begin{cases} i & \text{if } i = 2^j,\ j \geq 1 \\ 1 & \text{otherwise} \end{cases}$$

Amortized cost over n operations:

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=1}^{\log n} 2^j = n + 2(n-1) = 3n - 2 < 3$$

Problem 17.2-2 (459): Redo previous problem using the accounting method.

Solution: Charge each operation three coins. If i is not a power of 2, use one coin to pay for the operation, and save two. When $i = 2^j$, the $2(2^j - 2^{j-1}) = 2(2^{j-1}) = 2^j$ saved coins pay for the operation.

Problem 17.3-2 (462): Redo previous problem using the potential method.

Solution: Let the potential after operation $i > 1$ be $\Phi_i = 2(i - 2^{\lfloor \log i \rfloor})$ and $\Phi_1 = 1$.

$i = 2^j > 1:\ \hat{c_i} = c_i + \Phi_i - \Phi_{i-1} = i + 2(i-i) - 2((i-1) - i/2) = 2.$

$2^j < i = 2^j + k < 2^{j+1}:\ \hat{c_i} = 1 + 2((2^j + k) - 2^j) - 2((2^j + k - 1) - 2^j) = 3.$

The amortized operation cost is ≤ 3.

Discussion of assignment 3

1. Start by sorting. Assume $n \leq m$.
2. Insertion and removal can have different amortized cost. Determine the lowest (constant) cost for both.