Practice exercises for the final exam

do not return

Exercise 1

You have decided to make a CD of your favourite songs. You already picked the songs, but you still need to decide in what order they should appear on the CD. Your task is to compute the "best" possible order for them to appear. For each pair of songs i and j you have chosen a "compatibility rating" $compat_{i,j}$, which is a positive number indicating how good you think b sounds when it is played right after i. $compat_{i,j}$ and $compat_{j,i}$ are generally *not* equal. Use dynamic programming to find an ordering of the songs which maximizes the sum of the compatibilities of consecutive songs. You do not need to print out the actual ordering, just the maximum total compatibility. For n songs your algorithm should run in $\mathcal{O}\left(n^2 2^n\right)$ time.

(a) List the table(s) your algorithm will use, and explain the meaning of each entry.

The table is $C[S,i]$, where S is a subset of the N songs and i is a member of S. $C[S,i]$ will be equal to the maximum total compatibility possible for the songs in set S, ending with song i.

(b) Specify the recurrence and the base case(s) used by your algorithm.

$$C[S,i] = \begin{cases} 0 & \text{if } |S| = 1 \\ \max_{k \in S \setminus \{i\}} (C[S \setminus \{i\}, k] + compat_{k,i}) & \text{if } |S| > 1 \end{cases}$$

(c) Implement you algorithm in pseudocode.

```
for i := 1 to n
    C[{i}, i] := 0
for |S| := 2 to n
        forallS with cardinality |S|
            foralli ∈ S
                C[S, i] := max_{k∈S\{i}}(C[S\{i}, k] + compat_{k,i})
return max_{1≤i≤n} C[{1..n}, i]
```

(d) Analyze the running time of your algorithm: justify your answer briefly.

The first loop takes n steps. The max inside the next loop nest runs in time $\mathcal{O}(|S|)$. Thus the "forall $i \in S$" loop runs in time $\mathcal{O}\left(|S|^2\right)$, or $\mathcal{O}\left(n^2\right)$ for any $|S|$. The two outer loops loop over all possible subsets $|S|$, i.e. 2^n. So the total running time is $\mathcal{O}\left(2^n n^2\right)$.

Exercise 2

120 students have preenrolled for "Algorithms and Data Structures" lecture. Each student must attend one of the five tutor sessions, and each tutor is can handle a group with at most 24 students. Each student has specified three choices for tutor. A "happiness" rating is assigned to each choice: assignment of a student to her/his first choice is worth 10, while the second choice is worth 7, and the third choice 3. Assignment of a student to a tutor not among her/his top three is worth 0. The problem is to determine an assignment of students to tutors that maximizes the total happiness of the enrolled students, without exceeding the capacity of any lab section. Express this problem as:

(a) a linear programming problem,

We define the input matrix of "happiness" constants:

$$h_{i,j} = \begin{cases} 10 & \text{if tutorial } j \text{ is the first preference of student } i \\ 7 & \text{if tutorial } j \text{ is the second preference of student } i \\ 3 & \text{if tutorial } j \text{ is the third preference of student } i \\ 0 & \text{if tutorial } j \text{ is not preferred by student } i \end{cases}$$

The variables:

$$x_{i,j} = \begin{cases} 1 & \text{if student } i \text{ is assigned to tutor } j \\ 0 & \text{otherwise} \end{cases}$$

Then the objective function will be:

$$\sum_{i=1}^{120} \sum_{j=1}^{5} x_{i,j} h_{i,j}$$

subject to:

$$\forall i \sum_{j=1}^{5} x_{i,j} = 1$$

$$\forall j \sum_{i=1}^{120} x_{i,j} = 24$$

(b) a min cost flow problem

We define the network with target t as follows:
$V = t \cup \{v_i | 1 \leq i \leq 120\} \cup \{w_i | 1 \leq i \leq 5\}$
edges (w_i, t) $1 \leq i \leq 5$ have capacity 24 and cost 0
edges (v_i, w_j) $1 \leq i \leq 120$ $1 \leq i \leq 5$ have capacity 1 and cost $-h_{i,j}$
$supply(v_i) = 1$ for $1 \leq i \leq 120$
$supply(w_i) = 0$ for $1 \leq i \leq 5$
$supply(t) = -120$

Exercise 3

Consider the Maximum Alternating Sum Subsequence (MASS) problem. Given a sequence $S = [x_1, x_2, \ldots, x_n]$ of positive integers, find the subsequence $A = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$, where $i_1 < i_2 < \cdots < i_k$, that maximizes the alternating sum $x_{i_1} - x_{i_2} + x_{i_3} - x_{i_4} + \cdots \pm x_{i_k}$.

For example, if $S = [4, 9, 2, 4, 1, 3, 7]$, the MASS is $A = [9, 2, 4, 1, 7]$ which evaluates to $9 - 2 + 4 - 1 + 7 = 17$. If $S = [7, 6, 5, 4, 3, 2, 1]$ the MASS is $A = [7]$. Clearly, the length of the MASS depends on the actual values in S . Assume that the length of S is at least one, and all its elements are integers greater than zero. The following questions ask you to develop a dynamic programming algorithm to find the value of the MASS (but not the actual subsequence) for a given sequence.

(a) List the table(s) your algorithm will use, and explain the meaning of each entry.

Let E and O be two one-dimensional tables with indices from zero to n. $O[i]$ will store the value of the maximum odd-length alternating sum subsequence of the first i elements of S, and $E[i]$ will store the value of the maximum even-length alternating sum subsequence of the first i elements of S.

(b) Specify the recurrence and the base case(s) used by your algorithm.

$$O(i) = \begin{cases} -\infty & \text{if } i = 0 \\ \max\{E(i-1) + x_i, O(i-1)\} & \text{if } |S| > 1 \end{cases}$$

$$E(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\{O(i-1) - x_i, E(i-1)\} & \text{if } |S| > 1 \end{cases}$$

(c) Implement you algorithm in pseudocode.

```
O[0] := −∞
E[0] := 0
for i := 1 to n
    O[i] := max{E[i − 1] + x_i, O[i − 1]}
    E[i] := max{O[i − 1] − x_i, E[i − 1]}
return O[n]
```

(d) Analyze the running time of your algorithm: justify your answer briefly.

If S would contain negative integers, the last line woul have to return $\max\{O[n], E[n]\}$. Since each iteration has a constant number of operations the algorithm run in $\mathcal{O}(n)$.

Exercise 4
The Euclidean TSP is to find a shortest cycle through n points in the plane. This problem differs from the general TSP problem in the sense that the interpoint distances are not arbitrary, but are inherited from the Euclidean metric. However, this is also NP-hard. We will design an $\mathcal{O}\left(n^3\right)$ time 1.5-approximation algorithm to it now.

- find a Euclidean MST T of these points ($\mathcal{O}(n \log n)$ time).
- find a minimum weight Euclidean matching M on the set X of vertices of odd degree in T (We can do this in $\mathcal{O}\left(n^3\right)$ time).
- find an Eulerian cycle ϕ on the graph $T \cup M$ and shortcut it skipping already visited points, obtaining tour ϕ'.

Show that the obtained graph is 1.5-approximation of the Euclidean TSP. Also argue why the cycle ϕ exists.

$|X|$ is even, therefore M is a perfect matching. Since edges of M are adjacent only to the odd degree vertices in T, after adding them to T, all the nodes of the resulting graph $T \cup M$ have even degrees. Therefore this graph is Eulerian and cycle ϕ exists.
Weight of T is smaller than the weight of the optimal TSP τ:

$$W(T) < W(\tau)$$

Let us consider the optimal TSP τ' on the set V' of odd degree vertices of T. Since $|V'|$ is even, tour τ' consists of two perfect matchings on V' (M_1 and M_2), such that $W(M_1) + W(M_2) = W(\tau')$. Then:

$$W(M) \leq \min\{W(M_1), W(M_2)\} \leq \frac{W(M_1) + W(M_2)}{2} = \frac{W(\tau')}{2} < \frac{W(\tau)}{2}$$

The last inequality is true, because $|V| > |V'|$.
Finally, combining the results $W(T) < W(\tau)$ and $W(M) < W(\tau)/2$ and the fact that shortcutting with Euclidean distances can not increase the weight of a TSP tour, we get:

$$W(\phi') \leq W(\phi) = W(T) + W(M) < 1.5W(\tau)$$

Exercise 5
Given a simple[1] polygon P and a point a, we want to know whether a is inside P.

(a) Let R be any ray originating at a. Show that a is inside P iff R intersects an odd number of P's edges.

(b) Describe a linear time algorithm that finds whether a is inside P or not.

The problem of, given a polygon and a point, determining whether the point is inside the polygon. This can be done as follows:

```
FUNCTION insidePolygon(P: Polygon, p: Point) : Boolean
    ray : LineSegment := (p) − −(∞, ∞)
    counter := 0
    for i := 0 to P.length − 1 do
        if p ∈ p[i] − −p[i + 1] then
            return true
            x : Point := intersection(p[i] − −p[i + 1], ray)
            if x! = nil ∧ x! = p[i + 1] then
                if x = p[i] ∧
                    ((p[i + 1] right of x ∧ p[i − 1] right of x) ∨
                    (p[i + 1] left of x ∧ p[i − 1] left of x)) then
                    // We touched a corner, don't do anything
                else counter := counter + 1
    return mod(counter, 2) = 1
```

This algorithm uses a ray originating at the point p. It counts the number of intersections of this ray with the polygon boundary. If we imagine sitting on the ray, then crossing a boundary means exiting the polygon or entering the polygon. In the very end, we will be outside the polygon, because the ray is infinitely long. Hence, by looking back on how often we crossed the boundary, we can determine whether the origin was inside or outside the polygon, depending on whether we crossed the boundary an even or an odd number of times. While doing so, we have to take care with the corners (i.e. polygon vertices): If the ray is for instance outside the polygon, it may just touch such a corer and stay outside. This case appears when the ray hits a polygon vertex and the successor and predecessor of this vertex lie on the same side of the ray. We simply do not count intersections of this kind.

Exercise 6
You are given a city with n post offices. Your goal is to find a place for a new post office as far as possible from the existing post offices, but it must be inside the convex hull of all the offices. Design an algorithm for solving this problem that runs in $\mathcal{O}(n \log n)$ time.

Use the solution for Exercise 10.3 to find a largest empty circle that contains no post offices in its interior. The center of this circle is the place for the new post office.

Exercise 7
Do not forget to make the evaluation of this course: `www.st.cs.uni-sb.de/eva` using the password given to you at the lecture. If do not have a password, send a request to `dementiev@mpi-sb.mpg.de`. You feedback is highly appreciated.

[1] A polygon is simple if its boundary does not intersect with itself.