Contents

1 Homework Assignment Sheet I (due April 4, 2005) 2
2 Homework Assignment Sheet II (due April 11, 2005) 5
3 Homework Assignment Sheet III (due April 18, 2005) 10
4 Homework Assignment Sheet IV (due April 25, 2005) 18
5 Homework Assignment Sheet V (due May 9, 2005) 21
6 Homework Assignment Sheet VI (due May 23, 2005) 25
7 Homework Assignment Sheet VII (due June 06, 2005) 29
8 Homework Assignment Sheet VIII (due June 13, 2005) 33
9 Additional material 35
1 Homework Assignment Sheet I (due April 4, 2005)

Assignment 1 Consider $n$ points $P_1, P_2, \ldots, P_n \in \mathbb{R}^2$, and denote the vector from the origin to point $P_i$ by $\vec{p}_i$. Does the set of all convex combinations of $\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_n$ form a convex set? Prove or disprove this conjecture.

Proof. (By contradiction) Suppose the set $S$ of all possible convex combinations of $\vec{p}_1, \ldots, \vec{p}_n$ is not convex. Then there exist a segment connecting two points $Q, R \in S$ that does not lie entirely in $S$, that is there is a point $T \in \overline{PQ}$ such that $T \notin S$. As the points $Q, R$ are in $S$ we can write them as convex combinations of $P$:

$$\vec{q} = \sum_{i=1}^{n} \lambda^q_i \vec{p}_i, \quad \forall (1 \leq i \leq n) \lambda^q_i \geq 0, \quad \sum_{i=1}^{n} \lambda^q_i = 1$$

$$\vec{r} = \sum_{i=1}^{n} \lambda^r_i \vec{p}_i, \quad \forall (1 \leq i \leq n) \lambda^r_i \geq 0, \quad \sum_{i=1}^{n} \lambda^r_i = 1$$

And the point $T \in \overline{PQ}$ has form:

$$\vec{t} = \alpha \vec{q} + (1 - \alpha) \vec{r}, \quad 0 \leq \alpha \leq 1$$

We can then easily rewrite $\vec{t}$ as a linear combinations of points in $P$ by unfolding and refolding as appropriate the summations:

$$\vec{t} = \alpha \vec{q} + (1 - \alpha) \vec{r} =$$

$$= \alpha \sum_{i=1}^{n} \lambda^q_i \vec{p}_i + (1 - \alpha) \sum_{i=1}^{n} \lambda^r_i \vec{p}_i =$$

$$= \sum_{i=1}^{n} \alpha \lambda^q_i \vec{p}_i + \sum_{i=1}^{n} (1 - \alpha) \lambda^r_i \vec{p}_i =$$

$$= \sum_{i=1}^{n} \left[ \alpha \lambda^q_i + (1 - \alpha) \lambda^r_i \right] \vec{p}_i$$

$$= \sum_{i=1}^{n} \lambda^*_i \vec{p}_i$$

But this combination is convex:

$$\lambda^*_i = \alpha \lambda^q_i + (1 - \alpha) \lambda^r_i \geq 0$$
\[ \sum_{i=1}^{n} \lambda_i^T = \sum_{i=1}^{n} \left[ \alpha \lambda_i^T + (1 - \alpha) \lambda_i^F \right] = \alpha \sum_{i=1}^{n} \lambda_i^T + (1 - \alpha) \sum_{i=1}^{n} \lambda_i^F = \alpha + (1 - \alpha) = 1 \]

We have thus contradicted that \( T \notin S \) and \( S \) is then indeed convex!

Additional notes: one can prove also that the set \( S \) is the convex hull of \( P \) (de Berg et al., 1997, p.236 and Ex.11.1 p.249).

**Assignment 2** Prove or disprove: the intersection of a finite number of convex sets in \( \mathbb{R}^n \) forms a convex set.

We start by proving that the intersection of two convex sets is convex.

**Theorem 2.1.** Given two convex sets \( S, T \), \( S \cap T \) is convex.

*Proof.* (By contradiction) Suppose \( S \cap T \) is not convex. Then there exist \( s, t \in S \cap T \) such that there is a \( u \in \mathbb{R}^n, u \notin S \cap T \). But if \( u \notin S \cap T \), it can only be one of these cases:

- \( u \notin S \): this is impossible because \( s, t \in S \) and for convexity of \( S \), \( u \in S \).
- \( u \notin T \): this is impossible because \( s, t \in T \) and for convexity of \( T \), \( u \in T \).
- \( u \notin S \) and \( u \notin T \): this is impossible from the previous cases.

As such a \( u \) does not exist, we have a contradiction: \( S \cap T \) is then convex.

We can now easily prove the assignment.

**Theorem 2.2.** Given \( m \) convex sets \( S_1, \ldots, S_m \subseteq \mathbb{R}^n \), \( \bigcap_{i=1}^{m} S_i \) is convex.

*Proof.* (By induction on \( m \))

- \( m = 1 \): Trivial.
- \( m = 2 \): True, for theorem 2.1.
- \( m > 2 \): By induction hypothesis, the intersection of \( \bigcap_{i=1}^{m-1} S_i \) is a convex set. By theorem 2.1, also \( \bigcap_{i=1}^{m-1} S_i \cap S_m = \bigcap_{i=1}^{m} S_i \) is convex.
Assignment 3 Consider a planar graph, count the number of its vertices that are of odd degree, and let $k$ be that number. Prove or disprove: the number $k$ is always even.

We denote the degree of vertex $v$ as $\delta(v)$.

**Theorem 3.1.** Given a graph $G = (V, E)$, we have $\sum_{v \in V} \delta(v) = 2|E|$

*Proof.* In counting the sum of the degrees of each vertex in the graph, we count every edge exactly twice (one for each of its ends), that leads to the given formula. \(\square\)

**Theorem 3.2.** Given a graph $G = (V, E)$, the number of vertices in $V$ that are of odd degree is even.

*Proof.* By theorem 3.1 $\sum_{v \in V} \delta(v) = 2|E|$. We partition now the set $V$ into the set of vertices with odd degree $V_1$ and the set of vertices with even degree $V_2$. We can decompose the previous summation in:

$$\sum_{v \in V} \delta(v) = 2|E| = \sum_{v \in V_1} \delta(v) + \sum_{v \in V_2} \delta(v)$$

The second term is clearly even, being the summation of even numbers. As the element to the right of the equation is even, then also the first term of the summation (the sum of degrees of vertices with odd degree) has to be even. But a sum of $k$ odd numbers can be even only if $k$ is even, so $|V_1|$ is necessarily even. \(\square\)

Additional notes: the proved theorem weakens the original assignment hypothesis, as the planarity of the graph is not used. The first proof of these two theorems is attributed to Euler (Diestel, 1997, Prop. 1.2.1 p.5).
2 Homework Assignment Sheet II (due April 11, 2005)

Assignment 4 The algorithm presented in class for constructing a k-D tree of n points makes use of a linear-time algorithm for median finding. Replace the median-finding algorithm by an algorithm that is easier to implement in practice, while still maintaining the overall $O(n \log n)$ upper bound on the complexity of the construction.

An algorithm for the selection of the median in linear time is actually not so easy to implement, also if such an algorithm is known from the seventies (Cormen et al., 2001, Section 9.3 pp.189-192).

For the task of constructing k-D trees there is anyway an easier implementation solution (de Berg et al., 1997, Section 5.2 pp.99-105): we can presort the list of points on each of the k coordinates, and use this information to find the median in $O(1)$ time and to prepare the sorted lists for the next call in $O(n)$ time. More precisely, the total time complexity this way results in:

$$T(n) = T_p(n) + T_b(n)$$

Where $T_p(n)$ is the preprocessing time and $T_b(n)$ the time to build the k-D tree. The preprocessing time is given by sorting the points for all the coordinates and it is thus:

$$T_p(n) = k \ O(n \log n)$$

The k-D tree building time is given by the following recursion equation, where we equally split the input and do the $O(n)$ operation of partitioning the sorted lists into two sorted lists for the next call:

$$T_b(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
2T_b \left( \frac{n}{2} \right) + \Theta(n) & \text{otherwise}
\end{cases}$$

This recurrence equation can be solved with the master theorem, yielding the $O(n \log n)$ bound. The total complexity, assuming $k$ constant, is then:

$$T(n) = T_p(n) + T_b(n) = O(n \log n) + O(n \log n) = O(n \log n)$$

Keeping the “generic position assumption”, the following is the pseudocode implementing the proposed algorithm.

BuildKdTree($P, k$)
1. for $i = 1$ to $k$
2. do ▷ Sort the set $P$ on coordinate $i$
3. $SP[i] \leftarrow $ Sort($P, i$)
4. return BuildKdTreeR($SP, 0$)
\textbf{BuildKdTreeR}(SP, p, q, depth)

1. \( d \leftarrow \text{depth mod } k + 1 \)
2. \textbf{if} \( p \geq q \) \textbf{then return} \textbf{Nil} \hspace{2cm} \triangleright \text{Sentinel}
3. \textbf{elseif} \( q - p = 1 \)
4. \( m \leftarrow \text{SortedPartition}(SP, p, q, d) \)
5. \begin{align*}
\text{left} & \left[ x \right] \leftarrow \text{BuildKdTreeR}(SP, p, m - 1, \text{depth} + 1) \\
\text{right} & \left[ x \right] \leftarrow \text{BuildKdTreeR}(SP, m + 1, q, \text{depth} + 1)
\end{align*}
6. \textbf{return} \( x \) \hspace{2cm} \triangleright \text{Node}

\textbf{SortedPartition}(SP, p, q, d)

1. \( m \leftarrow \lfloor (p + q)/2 \rfloor \)
2. \( r \leftarrow SP[d][m][d] \)
3. \textbf{for} \( h = 1 \) \textbf{to} \( k \)
4. \begin{align*}
\textbf{do} \ i & \leftarrow p \\
\textbf{do} & \ j \leftarrow m + 1 \\
\textbf{while} & \ i < m \text{ and } j \leq q
\end{align*}
5. \begin{align*}
\textbf{do} & \ \textbf{if} \ SP[h][i][d] > r \text{ and } SP[h][j][d] \leq r \\
& \ \textbf{then exchange} \ A[i] \leftrightarrow A[j] \\
& \ \textbf{if} \ SP[h][i][d] \leq r \\
& \ \textbf{then} \ i \leftarrow i + 1 \\
& \ \textbf{if} \ SP[h][j][d] > r \\
& \ \textbf{then} \ j \leftarrow j + 1
\end{align*}
6. \textbf{return} \( m \)
Assignment 5  Given are two sets \( A := \{a_1, a_2, \ldots, a_n\} \) and \( B := \{b_1, b_2, \ldots, b_n\} \) of real numbers. How many pairwise comparisons among elements of the sets are necessary in the worst case for a comparison-based algorithm to check whether \( A = B \)?

The problem is known in the literature as **Set equality**, and proposed as a problem with difficulty M31 in Knuth’s “Art of Computer Programming” (Knuth, 1973, Section 5.3.2, ex.23, p.207). It is curious to note that, although the bound with comparisons is \( \Omega(n \log n) \), in case the numbers are integers and multiplications are allowed, the bound can be lowered to \( \Omega(n) \).

We suppose that all the comparisons are equality checks between one element of \( A \) and one of \( B \), and we devise an adversary that forces \( n(n+1)/2 \) such comparisons.

The adversary \( E \) builds a matrix \( M_{n \times n} \), where each \( m_{i,j} \in \{0, 1, 2\} \), and initialize the matrix to all zeros. This is the behaviour of \( E \) when the set equality algorithm \( S \) performs a comparison \( A_i = B_j \), described as an algorithm:

\[
\begin{align*}
E(i,j) & \quad 1 \text{ if } M[i,j] \neq 2 \\
& \quad 2 \text{ then } M[i,j] \leftarrow 1 \\
& \quad 3 \text{ if } M[i, \ldots] \text{ or } M[\ldots, j] \text{ contains a two} \\
& \quad 4 \text{ then } \text{return } M[i,j] = 2 \\
& \quad 5 \text{ if } M[i, \ldots] + M[\ldots, j] \geq n - 1 \\
& \quad 6 \text{ then } M[i,j] = 2 \\
& \quad 7 \text{ return True} \\
& \quad 8 \text{ else return False}
\end{align*}
\]

Intuitively, \( E \) is adjusting the permutation to make it difficult for \( S \) to guess. The answers of \( E \) are consistent as it keeps tracks of elements said to be equal (the twos in the matrix) and it always says that an element in \( A \) is equal to only another element of \( B \) (and viceversa), thus avoiding to have to deal with transitivity. The minimum number of non-zero elements that are present after a set equality algorithm correctly terminated against \( E \) gives the minimum number of comparisons that \( S \) has to perform, as they are comparisons that \( S \) did at least once (only once if \( S \) is optimal). We can notice that if \( S \) is correct it cannot give up saying that the sets are not equal until it checked all the possible permutations. The number of twos in \( M \) gives the size of the permutation currently found (more precisely, the number of permutations missing is between \((n-m-1)! \) and \((n-m)!\), where \( m \) is the number of twos in \( M \)). Notice that a correct \( S \) has to produce a permutation at the end, if playing against \( E \). We can observe that:

- the algorithm \( S \) thus has to produce \( n \) twos in \( M \) to terminate correctly;
• for the way $E$ is built, in $M$ there is only a two per row and only a two per column;

• for the way $E$ is built, in $M$ for each two there at least $n - 1$ ones in its same row and column;

• so each one in $M$ has a two in the same row and a two in the same column.

The minimum number of comparisons is given by:

$$\frac{n(n - 1)}{2} + \frac{n(n + 1)}{2} = \Omega(n^2)$$

It is simple to write an algorithm that performs $\Omega(n^2)$ comparisons, that proves that the adversary is optimal.

**Assignment 6** Consider $n$ real numbers $x_1, x_2, \ldots, x_n$ which are given in an arbitrary (unsorted) order. Applying the operation $MG$ to those numbers shall yield the maximum gap in the sorted list of those numbers (thus, if $y_1, y_2, \ldots, y_n$ is the sorted order of $x_1, x_2, \ldots, x_n$ then $MG(x_1, x_2, \ldots, x_n) := \max_{1 < i < n} (y_{i+1} - y_i)$). The decision operation $UG$ applied to $x_1, x_2, \ldots, x_n$ will yield true if and only if there exists some constant $\delta$ such that $y_{i+1} = y_i + \delta$ for $1 \leq i < n$ (the constant $\delta$ need not be determined by $UG$, though). Does the knowledge of a lower bound on the minimum complexity of $MG$ help to deduce a lower bound on the minimum complexity of $UG$? Conversely, does the knowledge of a lower bound on the minimum complexity of $UG$ help to deduce a lower bound on the minimum complexity of $MG$? Why (not)?

The problem **UNIFORM GAP** is linear time transformable to **MAX GAP**, as shown by the following algorithm:

**UNIFORM Gap** ($A$)
1. $a_n \leftarrow \text{MAX}(A)$
2. $a_1 \leftarrow \text{MIN}(A)$
3. $\delta \leftarrow \text{MAX Gap}(A)$
4. if $(a_n - a_1)/\delta = \text{length}[A] - 1$
   then return TRUE
6. else return FALSE

It is easy to see that the complexity of this algorithm without the call to $\text{MAX GAP}$ is linear, as $\text{MIN}$ and $\text{MAX}$ can be performed in linear time and the rest in constant time.

For seeing the correctness we prove the following theorem.
Theorem 6.1. Let $A$ be a set with a total ordering relation. Then
\[
\frac{\max(A) - \min(A)}{MG(A)} = |A| - 1 \iff UG(A) = \text{true}
\]

Proof. Without loss of generality, we consider the elements of $A$ ordered $A = \{a_1, \ldots, a_n\}, a_1 \leq \ldots \leq a_n$. We can then rewrite the left side as:
\[
\frac{a_n - a_1}{MG(A)} = n - 1
\]

$(\Leftarrow)$ If $UG(A) = \text{true}$ then all the gaps in $A$ are the same $a_{i+1} - a_i = \delta$ for $1 \leq i < n$. Then $a_n - a_1 = \delta(n - 1)$, but as $MG(A) = \delta$ we obtain $a_n - a_1 = MG(A)(n - 1)$ as required.

$(\Rightarrow)$ (By contradiction) Suppose $UG(A) = \text{false}$ then there is at least a $i, 1 \leq i < n$, such that $a_{i+1} - a_i < MG(A)$ (as at least one gap is $MG(A)$ and they cannot be all equal gaps for the contradiction hypothesis). But then this holds, as at least one term is less than $MG(A)$ and no term is bigger than $MG(A)$:
\[
\sum_{i=1}^{n-1} a_{i+1} - a_i < MG(A)(n - 1)
\]
\[
a_n - a_1 < MG(A)(n - 1)
\]

But this last contradicts our hypothesis.

Thus the knowledge of a lower bound on Uniform Gap transfers also to the Max Gap problem in this way:
\[
T_{MG} = \Omega(T_{UG} - n)
\]
3 Homework Assignment Sheet III (due April 18, 2005)

Assignment 7 Show that storing all edges of all chains may indeed cause the “chain method” to consume $O(n^2)$ space, where $n$ denotes the number of input vertices of the PSLG.

Given an $n$, we build a PSLG with $n$ vertices such that storing all edges of all chains in the “chain method” would consume $O(n^2)$ space. Without loss of generality, consider $n = 4m$ for some $m \in \mathbb{N}$. We then build the following PSLG:

Note that the chain method on this PSLG must create $\frac{n}{2}$ chains for the nodes in the middle, and that each chain has a number of edges that is (counting from the external chain) increasing; more precisely, the total amount of space required to store all the edges is:

$$2 \sum_{i=1}^{\frac{n}{4}} 2i = 4 \sum_{i=1}^{\frac{n}{4}} i = \frac{n(n + 4)}{8} = \Theta(n^2)$$

Another slightly more complicated worst-case PSLG can be found in Preparata and Shamos’ book (Preparata and Shamos, 1985, Figure 2.16(b), p. 54).
Assignment 8  Consider an $n$-vertex polygonal chain

\[ C := \{p_1, p_2, \ldots, p_n\}. \]

Devise an $O(n)$ algorithm:

1. to check whether $C$ is monotone with respect to some line, and
2. to compute such a line if it exists.

Given a chain $C$ we can observe that there is a relation between the normals of the edges in the chain and the possible directions of monotony: in particular, each acute angle between subsequent edges defines a set of directions with respect to which the chain cannot be monotone.

If we take then the set of all possible directions $[0, \pi)$ and we remove the set of directions not allowed by each couple of consequent edges, we have an algorithm that checks the monotonicity (by seeing if the final set is empty) and in case the chain is monotone gives the set of possible directions of monotonicity.

We developed an algorithm using intervals to represent the set of possible directions. In truth the algorithm computes the set of directions not allowed, that lies in a single interval, so it is represented by the extremes. It is easy to check the the set of directions not allowed is in a single interval:

- taking two consecutive edges $i - 1$ and $i$, its set of disallowed directions starts from the normal of $i$ and goes towards the normal of $i - 1$ clockwise or counterclockwise, depending on which one is the acute angle (the smaller set);
- the edge $i - 1$ has to be in the interval of disallowed directions;
- then adding this set of disallowed directions can only increase the interval.

By analyzing all the possible relations between the extremes of the disallowed interval, the normal of $i - 1$ and the normal of $i$, we can come up with a linear algorithm.

The following procedure returns (direction is in degrees from 0 to 180):

- $(0, 180)$ if the chain is not monotone;
- $(a, a)$ if the chain is strictly monotone for $[0, \pi) \setminus a$;
- $(a, b)$ if the chain is strictly monotone for these directions:
  - if $a < b$ for every direction in $(a, b)$;
  - if $a > b$ for every direction in $[0, b) \cup (a, 180]$.

\[
\text{Area2}(p_1, p_2, p_3) \\
1 \quad \text{return} \ (p_2.x - p_1.x)(p_3.y - p_1.y) - (p_3.x - p_1.x)(p_2.y - p_1.y)
\]
Figure 1: This figure shows the cases for left turns in the algorithm **MonotoneDirection**.
Figure 2: This figure shows the cases for right turns in the algorithm \textsc{MonotoneDirection}.
\texttt{LEFT}(p_1, p_2, p_3)
1 \textbf{return} \ \text{Area2}(p_1, p_2, p_3) > 0

\texttt{NORM}(p_1, p_2)
1 \textbf{return} \ \text{ATAN2}(p_2.y - p_1.y, p_2.x - p_1.x) + 90 \mod 180

\texttt{MONOTONE \! DIRECTION}(C)
\begin{itemize}
\item Precondition: \texttt{length}[C] > 1
\item Precondition: no collinear subsequent points in \texttt{C}
\end{itemize}
1 \quad \texttt{min} \leftarrow \text{max} \leftarrow \text{NORM}(C[1], C[2])
2 \quad n \leftarrow \text{length}[C]
3 \quad \textbf{for} \ i = 3 \ \textbf{to} \ n
4 \quad \textbf{do} \ n_i \leftarrow \text{NORM}(C[i-1], C[i])
5 \quad \quad n_{i-1} \leftarrow \text{NORM}(C[i-2], C[i-1])
6 \quad \quad \textbf{if} \ \texttt{LEFT}(C[i-2], C[i-1], C[i])
7 \quad \quad \quad \textbf{then} \triangleright \ \text{Left turn}
8 \quad \quad \quad \quad \textbf{if} \ \texttt{max} \geq \texttt{min} > n_i \ \text{or}
9 \quad \quad \quad \quad \quad n_i > \texttt{max} \geq \texttt{min} \ \text{or}
10 \quad \quad \quad \quad \quad \texttt{min} > n_i > \texttt{max}
11 \quad \quad \quad \quad \quad \textbf{then} \triangleright \ \text{Reduces the possible monotony directions}
12 \quad \quad \quad \quad \quad \texttt{max} \leftarrow n_i
13 \quad \quad \quad \ \textbf{elseif} \ \texttt{max} \geq n_{i-1} > n_i \geq \texttt{min} \ \text{or}
14 \quad \quad \quad \quad \quad \texttt{min} > \texttt{max} \geq n_{i-1} > n_i \ \text{or}
15 \quad \quad \quad \quad \quad n_i \geq \texttt{min} > \texttt{max} \geq n_{i-1} \ \text{or}
16 \quad \quad \quad \quad \quad n_{i-1} > n_i \geq \texttt{min} > \texttt{max}
17 \quad \quad \quad \quad \quad \textbf{then} \ \textbf{return} \ (0, 180)
18 \quad \quad \quad \ \textbf{else} \triangleright \ \text{Right turn}
19 \quad \quad \quad \quad \textbf{if} \ \texttt{max} \geq \texttt{min} > n_i \ \text{or}
20 \quad \quad \quad \quad \quad n_i > \texttt{max} \geq \texttt{min} \ \text{or}
21 \quad \quad \quad \quad \quad \texttt{min} > n_i > \texttt{max}
22 \quad \quad \quad \quad \quad \textbf{then} \triangleright \ \text{Reduces the possible monotony directions}
23 \quad \quad \quad \quad \quad \texttt{min} \leftarrow n_i
24 \quad \quad \quad \ \textbf{elseif} \ \texttt{max} \geq n_{i-1} > n_i \geq \texttt{min} \ \text{or}
25 \quad \quad \quad \quad \quad \texttt{min} > \texttt{max} \geq n_{i-1} > n_i \ \text{or}
26 \quad \quad \quad \quad \quad n_{i-1} \geq \texttt{min} > \texttt{max} \geq n_i \ \text{or}
27 \quad \quad \quad \quad \quad n_i > n_{i-1} \geq \texttt{min} > \texttt{max}
28 \quad \quad \quad \quad \quad \textbf{then} \ \textbf{return} \ (0, 180)
29 \quad \textbf{return} \ (C_b, C_t)

It is easy to check that the procedure runs in optimal $O(n)$ time.
Assignment 9  A polygon \( P \) is called \( x \)-monotone if it can be partitioned into two polygonal chains that are monotone relative to the \( x \) axis. How efficiently can repetitive-mode point-in-polygon queries be performed on \( x \)-monotone polygons? How much time needs to be spent on the preprocessing? (As usual, the term “polygon” refers to both the closed polygonal chain \( C \) which is to be partitioned into two monotone chains, as well as to the region bounded by \( C \).)

If \( P \) can be partitioned in two polygonal chains monotone with respect to the \( x \) axis, it means that:

- the starting and ending points of the chains are the same (if they are stored with respect to the monotocity);
- for the two chains to be monotone with respect to the \( x \) axis, their starting point must be the leftmost point in \( P \), and their ending point the rightmost point in \( P \);
- we can distinguish a top and a bottom chain, as all the points lying on the polygonal line of a chain must be over the same \( x \) coordinate point on the other chain (otherwise the polygon would be degenerate).

Once noticed this, the chains can be found in \( O(n) \) time by finding the leftmost and rightmost points, splitting the polygon, and reverting the order of the points of the superior chain (supposing \( P \) is stored CCW).

Doing a point-in-polygon query would then require \( O(\log n) \) time, as it would be sufficient a binary search on the top and bottom chain for finding the segment where the point lies and to check if the point is below the top line and above the bottom line.

The algorithm for the \( O(n) \) preprocessing is the following:
\textbf{SplitPolygon}(P)

1. \text{\textit{n} $\leftarrow$ length}[P]
2. \text{\textit{min} $\leftarrow$ 1}
3. \text{\textit{max} $\leftarrow$ 1}
4. \textbf{for} \text{\textit{i} = 1 \textbf{to} \textit{n}}
5. \hspace{1em} \textbf{do if} \text{\textit{P}[i].x < \textit{P}[min].x}
6. \hspace{2em} \text{\textit{min} $\leftarrow$ \textit{i}}
7. \hspace{1em} \textbf{if} \text{\textit{P}[i].x > \textit{P}[max].x}
8. \hspace{2em} \text{\textit{max} $\leftarrow$ \textit{i}}
9. \hspace{1em} \textbf{if} \text{\textit{min} < \textit{max}}
10. \hspace{2em} \text{\textit{C}_{b} $\leftarrow$ \textit{P}[\textit{min} \ldots \textit{max}]
11. \hspace{2em} \text{\textit{C}_{t} $\leftarrow$ \text{\textit{REVERSE}}(\textit{P}[1 \ldots \textit{min}] \cup \textit{P}[\textit{max} \ldots \textit{n}])}
12. \hspace{2em} \text{\textit{else} \textit{C}_{b} $\leftarrow$ \textit{P}[\textit{min} \ldots \textit{n}] \cup \textit{P}[1 \ldots \textit{max}]
13. \hspace{2em} \text{\textit{C}_{t} $\leftarrow$ \text{\textit{REVERSE}}(\textit{P}[\textit{max} \ldots \textit{min}])}
14. \textbf{return} (\textit{C}_{b}, \textit{C}_{t})

The algorithm for a \textit{O}(log \textit{n}) query is this:

\textbf{Area2}(p_{1}, p_{2}, p_{3})

1. \textbf{return} \((p_{2}.x - p_{1}.x)(p_{3}.y - p_{1}.y) - (p_{3}.x - p_{1}.x)(p_{2}.y - p_{1}.y)\)

\textbf{LeftOn}(p_{1}, p_{2}, p_{3})

1. \textbf{return} \textbf{Area2}(p_{1}, p_{2}, p_{3}) \geq 0

\textbf{BinarySearch}(p, C, l, r)

1. \textbf{if} \textit{r} = \textit{l} + 1
2. \hspace{1em} \textbf{then return} \textit{l}
3. \hspace{1em} \textit{m} $\leftarrow$ \lfloor(l + r)/2\rfloor
4. \hspace{1em} \textbf{if} \textit{p}.x \leq \textit{C}[\textit{m}].x
5. \hspace{2em} \textbf{then} \textbf{BinarySearch}(\textit{p}, \textit{C}, \textit{l}, \textit{m})
6. \hspace{2em} \textbf{else} \textbf{BinarySearch}(\textit{p}, \textit{C}, \textit{m} + 1, \textit{r})
PointInPolygon(p, \mathcal{C}_b, \mathcal{C}_t)

1. \( n_b \leftarrow \text{length}[\mathcal{C}_b] \)
2. \( n_t \leftarrow \text{length}[\mathcal{C}_t] \)
3. \( \triangleright \) Check if the coordinate \( x \) is in the polygon:
4. \( \text{if } p.x < \mathcal{C}_b[1].x \text{ or } p.x > \mathcal{C}_b[n_b].x \)
5. \( \quad \text{then return False} \)
6. \( \triangleright \) Check if it is above the bottom chain:
7. \( i \leftarrow \text{BinarySearch}(p, \mathcal{C}_b, 1, n_b) \)
8. \( \text{if not LEFTON}(p, \mathcal{C}_b[i], \mathcal{C}_b[i + 1]) \)
9. \( \quad \text{then return False} \)
10. \( \triangleright \) Check if it is under the top chain:
11. \( i \leftarrow \text{BinarySearch}(p, \mathcal{C}_t, 1, n_t) \)
12. \( \text{if not LEFTON}(p, \mathcal{C}_t[i], \mathcal{C}_t[i + 1]) \)
13. \( \quad \text{then return False} \)
14. \( \text{return True} \)
Assignment 10  Consider a simple polygon $\mathcal{P}$ and a set $\mathcal{S}$ of points that belong to $\mathcal{P}$. (As usual, the term “polygon” refers to the union of the closed polygonal chain $\mathcal{C}$ and the region bounded by $\mathcal{C}$.) We say that $\mathcal{P}$ is surveilled by $\mathcal{S}$ if for every point $P \in \mathcal{P}$ there exists a point $S \in \mathcal{S}$ such that the line segment $PS$ is contained in $\mathcal{P}$. Thus, every $P \in \mathcal{P}$ has to be seen by at least one $S \in \mathcal{S}$.

1. Let $\mathcal{S}$ be the set of all mid-points of edges of (the boundary of $\mathcal{P}$). Is this set $\mathcal{S}$ always sufficient to surveil $\mathcal{P}$?

2. Let $\mathcal{S}$ be the set of all convex vertices of $\mathcal{P}$ (a vertex is convex if its interior angle is less than $180^\circ$). Is this set $\mathcal{S}$ always sufficient to surveil $x$-monotone polygons?

Both propositions are false, as shown by the following two counterexamples (the shaded dark gray area is a set of points not visible by $\mathcal{S}$):

Assignment 11  Consider a convex polygon $\mathcal{P}$ and two points $P, Q \in \mathcal{P}$. Prove: if the distance between $P$ and $Q$ is maximum among all pairs of points of $\mathcal{P}$ then both $P$ and $Q$ are vertices of $\mathcal{P}$.

Proof. (By contradiction) Suppose $P$ and $Q$ are not vertices of $\mathcal{P}$. Then, we can have one of the two cases:

- Both points are on the boundary of the polygon. But then, if we take a point $Q'$ on the same edge where $Q$ lies in the direction where $PQ$ forms
with the edge an angle bigger or equal than 90 degrees, the point is in $\mathcal{P}$ and $d(P, Q') > d(P, Q)$.

- One of the two points is not on the boundary of the polygon (w.l.o.g. the point is $Q$). We can then prolong the segment $PQ$ in the $Q$ direction until we meet an edge in point $Q'$. But then, for every point $Q'' \neq Q$ in $QQ'$, the point is in $\mathcal{P}$ and $d(PQ'') > d(P, Q)$.

But both cases contradict the hypothesis that the distance between $P$ and $Q$ is maximum among all pairs of points of $\mathcal{P}$ and then $P$ and $Q$ must be vertices of $\mathcal{P}$. □

**Assignment 12** Suppose that you want to use interior elimination for speeding up CH-computations. So, first you use $min(x \pm y)$ and $max(x \pm y)$ to find four corners of a “large” rectangle $\mathcal{R}$, and second you inscribe the largest axis-parallel rectangle $\mathcal{R}'$ into $\mathcal{R}$, as explained in class. Show that for all $n \in \mathbb{N}$ there exists a set $S_n$ of $n$ points such $CH(S_n)$ has only four corners and $\mathcal{R}'$ contains no point of $S_n$ in its interior (you do not need to specify the coordinates of the points of $S_n$ explicitly, but it should be clear how you could come up with actual coordinates, and your reasoning has to be convincing, beyond the doubt associated with a sketchy hand-drawing).
Given an $n \in \mathbb{N}, n > 6$, we put 6 points in the following way:

![Diagram of points and their positions relative to each other, with axes labeled $x$ and $y$.]

We can now put the rest of the points in $\mathcal{R} \setminus \mathcal{R}'$: it is easy to check that this disposition of points satisfies all of the requirements.
Assignment 13 Consider a set $S$ of $n$ points in $\mathbb{R}^2$. A polygon $P$ is said to be a simple polygon on $S$ if $P$ is a simple polygon whose set of vertices is equal to $S$ (note that we do not require the points of $S$ to appear in some specific order in $P$). Devise a (worst-case) time-optimal algorithms for computing a simple polygon $P$ on $S$ such that

- $P$ is star-shaped;
- $P$ is $x$-monotone.

Note that you are requested to argue why the worst-case complexities of your algorithms are optimal!

**Theorem 13.1.** Building any simple polygon from a set $S$ of points requires $\Omega(n \log n)$ time.

**Proof.** `ConvexHull` and `BuildSimplePolygon` are easily reducible one to the other as the convex hull of $S$ is the same of the convex hull of any polygon with $S$ as vertices.

```latex
\begin{verbatim}
ConvexHull(S)
1  P ← BuildSimplePolygon(S)
2  return MelkmanConvexHull(P)
\end{verbatim}
```

But we know that Melkmans’s algorithm can find in $O(n)$ time the convex hull of a simple polygon, and as `ConvexHull` has an $\Omega(n \log n)$ lower bound, than this also transfers to `BuildSimplePolygon`. In fact, if an algorithm $A$ existed for `BuildSimplePolygon`, with $A = o(n \log n)$, then it could be used together with Melkmans’s algorithm to find a solution to `ConvexHull` in less than $\Omega(n \log n)$ time.

Any $O(n \log n)$ algorithm for building the polygons is then optimal.

1. To build a star-shaped polygon, we can use the sorting algorithm performed in the beginning of Graham’s scan, that runs in $O(n \log n)$ time.

2. To build an $x$-monotone polygon, we find the $x$ extremes of the polygon (if more than one, any of them will do), say $x_{\text{min}}$ and $x_{\text{max}}$, in $O(n)$ time. We then can split $S$ in points above and below $x_{\text{min}}, x_{\text{max}}$ in $O(n)$ time using determinant calculations, and we call the sets $S_a$ and $S_b$. Ordering the two sets by $x$ coordinate can be performed now in $O(n \log n)$ time and gives the required polygon ($S_b$ should be ordered from biggest to smallest, while $S_a$ from smallest to biggest). This gives the polygon in total $O(n \log n)$ time.
Assignment 14  We are again concerned with simple polygons on a set $S$ of $n$ points in $\mathbb{R}^2$, and attempt to construct a simple polygon on $S$ incrementally by “inserting” points of $S$ as new vertices into the polygon under construction: if $P' := (p_1, p_2, \ldots, p_k, p_1)$ is a simple polygon on $S' := \{p_1, p_2, \ldots, p_k\}$ as constructed so far, with $k < n$ and $S' \subset S$, then we

1. randomly pick a point $q \in S \setminus S'$,
2. look for a suitable edge $(p_i, p_j)$ of $P'$, and
3. replace that edge by the two edges $(p_i, q)$ and $(q, p_j)$, thus obtaining the new polygon $P'' := (p_1, p_2, \ldots, p_i, q, p_j, \ldots, p_k, p_1)$.

Of course, an edge $(p_i, p_j)$ is eligible for replacement only if the resulting polygon $P''$ is a simple polygon. This incremental construction is started on an initial 3-vertex polygon $(p_1, p_2, p_3, p_1)$, for three randomly chosen points $p_1, p_2, p_3$ of $S$. Will this algorithm always yield an $n$-vertex simple polygon on $S$ (no need to consider the time complexity of the algorithm; all we ask is whether this algorithm will always produce an $n$-vertex simple polygon on $S$)?

The answer is no. We show below a simple execution of this algorithm on a set of points that is unable to produce a simple polygon (in the last step, there is no edge eligible for replacement).
Assignment 15 In the lecture I claimed that all onion layers of a set $S$ of $n$ points in $\mathbb{R}^2$ can be computed in time $O(n \log n)$. And it is obvious how to obtain all onion layers in time $O(n^2 \log n)$. Your task is to compute all onion layers in time $O(n^2)$ (if it helps then you may assume that no three points of $S$ are collinear; of course, it does not suffice to simply refer to the classical $O(n \log n)$ algorithm!).

The algorithm was proposed by Shamos first and is presented in its book (Preparata and Shamos, 1985, sec. 4.2.1, p.173). The problem is called Depth of a Set and is solved simply using Jarvis’ march (the gift wrapping algorithm). We know that gift wrapping has complexity $O(nh)$ where $h$ is the number of vertices in the convex hull. We can then use the following algorithm:

\begin{verbatim}
ConvexLayers(S)
1  i ← 0
2  while S ≠ \emptyset
3      do S_i ← GiftWrapping(S)
4         S ← S \ S_i
5       i ← i + 1
\end{verbatim}
The complexity can be derived in the following way: for each point in each convex hull, at maximum \( O(n) \) operations have been performed (for the polar angle comparison); then, the total number of operations is \( O(n^2) \). A little more subtle reasoning is the following: every call to \textsc{GiftWrapping} requires \( O(nh) \) time, and if we sum all the \( h \) we obtain exactly \( n \); the recurrence relation is:

\[
T(n) = \begin{cases} 
O(1) & \text{if } n \leq 3 \\
O(nh) + T(n-h) & \text{if } n > 3 
\end{cases}
\]

where \( n = |S| \) and \( h = |CH(S)| \). The total complexity is then:

\[
\sum_{i=1}^{k} \left( n - \sum_{j=1}^{i-1} h_j \right) h_i, \quad \text{where } \sum_{i=1}^{k} h_i = n.
\]

\[
\sum_{i=1}^{k} \left( n - \sum_{j=1}^{i-1} h_j \right) h_i < \sum_{i=1}^{k} nh_i = n \sum_{i=1}^{k} h_i = n^2
\]

Another solution is the following, that does a \( O(n \log n) \) preprocessing and then performs the layering in linear time (I still have a doubt: if the layers are somehow proportional to \( n \), like the worst case that they all are triangles, are we sure that the following is still linear on the overall?):

**ConvexLayers'(S)**

1. \( S_x \leftarrow \text{SortOnX}(S) \)
2. \( i \leftarrow 0 \)
3. \textbf{while} \( S_x \neq \emptyset \)
4. \quad \textbf{do} \( S_i \leftarrow \text{DoubleGraham}(S) \)
5. \quad \quad \textgreater \text{ Sorted subtraction:}
6. \quad \quad \quad \quad S_x \leftarrow S_x \setminus S_i
7. \quad \quad \quad i \leftarrow i + 1
6 Homework Assignment Sheet VI (due May 23, 2005)

**Assignment 16** Consider a set $S$ of $n$ points in $\mathbb{R}^3$ and assume that their convex hull $CH(S)$ is known. For a plane $\epsilon$, let $S'$ be the set of points obtained by projecting all points of $S$ onto $\epsilon$. Does the knowledge of $CH(S)$ help to compute $CH(S')$ in $o(n \log n)$ time (you are welcome to assume that the points of $S$ are in general position)?

Without loss of generality, we assume the plan $\epsilon$ to be the $xy$ plan (if not, a simple rotation and translation suffice).

**Theorem 16.1.** Given a convex set $S \subseteq \mathbb{R}^3$, its projection $S'$ on the $xy$ plane is a convex set.

*Proof.* (By contradiction) Suppose $S'$ is not convex, then there are three points $p', q', r' \in \mathbb{R}^2$ such that $p', q' \in S'$ but $r' \notin S'$ and $r'$ is a convex combination of $p', q'$. But $p'$ and $q'$ are projections of some points $p$ and $q$ of $S$. Being $S$ convex, all the points in the convex combination of $p, q$ are in $S$ and it is easy to find a point $r$ such that its projection is exactly $r'$: but then $r' \in S'$ contradicting our assumption that $S'$ is not convex. $\square$

**Theorem 16.2.** Given a convex polyhedron $S \subseteq \mathbb{R}^3$, consider its convex polygon projection $S'$ on the $xy$ plane. Every edge of $S'$ is the projection of an edge of $S$ and every vertex of $S'$ is the projection of a vertex of $S$.

A convex hull in 3D is a polytope (a convex bounded region of space) for which the Euler formula is valid, so its number of edges is linear in the number of vertices. We can then come up with an $O(n)$ algorithm to solve the problem, whose steps are:

1. project $CH(S)$ to the $xy$-plane, maintaining the original edges;
2. use Jarvis’ march but visiting only neighboring vertices.

**Assignment 17** It is easy to see that the convex hull of $n$ points of $\mathbb{R}^2$ can be computed in $O(n)$ time if the points are given in sorted order according to their $y$-coordinates. Show that pre-sorting $n$ points of $\mathbb{R}^3$ according to their $z$-coordinates does not help to break the $\Omega(n \log n)$ lower bound for convex-hull computations in $\mathbb{R}^3$ (that is, computing the convex hull in $\mathbb{R}^3$ takes $\Omega(n \log n)$ even if one knows the sorted order of the points according to $z$-coordinates).
Proof. (By reduction from \textsc{ConvexHull2D}) \textsc{ConvexHull2D} can be linearly reduced to \textsc{ConvexHull3DSortedZ} by using the algorithm \textsc{ReduceConvexHull3dTo2d} in assignment 6:

\textsc{ConvexHull2D}(S)

1. \textbf{for} \( i \leftarrow 1 \) \textbf{to} \( \text{length}[S] \)
2. \textbf{do} \( S'[i] \leftarrow (S[i].x, S[i].y, i) \)
3. \( C \leftarrow \textsc{ConvexHull3DSortedZ}(S') \)
4. \textbf{return} \( \textsc{ReduceConvexHull3dTo2d}(C) \)

This means that, as \textsc{ReduceConvexHull3dTo2d} runs in linear time and the transformation is linear too, if \textsc{ConvexHull3DSortedZ} could be solved in time \( o(n \log n) \), than this bound would transfer to \textsc{ConvexHull2D}, that on the contrary we know has a \( \Omega n \log n \) time bound. \( \square \)
Assignment 18  In the lecture I claimed that the incremental construction of the Voronoi diagram of a set $S$ of $n$ points of $\mathbb{R}^2$ may result in the generation of $\Theta(n^2)$ many bisectors, for “appropriate” sets $S$ and “poorly” chosen insertion orders. Show that this claim is indeed true.

To show this claim we use the following example: we dispose the points along a spiral that does a single $2\pi$ turn and we insert them in the Voronoi diagram going along the spiral from the outer side toward the inner side. This forces, when inserting the $i$-th point, the computation of $(i-1)$ new bisectors, forcing a total $O(n^2)$ running time.

In the figure below the incremental Voronoi diagram is shown:

Note that $i-1$ bisectors are needed because the ray between the last three insert points inserted is always decreasing for the construction of the spiral.
This image represents the worst case incremental construction of Voronoi diagram with all Voronoi diagrams shown from light gray to dark gray in order of construction:

This is another case for worst case incremental construction of Voronoi diagram with all Voronoi diagrams shown from light gray to dark gray in order of construction. Inserting the horizontal points in any order and then all the $n^2$ vertical points from the bottom causes the computation of $\frac{n^2}{2}$ per vertical point:
Assignment 19  Consider a line-segment $pq$. A semi-circle over $pq$ is a circular arc with counter-clockwise orientation that is centered in the midpoint of $p$ and $q$ and that either starts in $p$ and ends in $q$ or starts in $q$ and ends in $p$. We say that a convex polygon has the semi-circle property if it has an edge $(p, q)$ such that all vertices other than $\{p, q\}$ lie within a semi-circle erected over $pq$. Show that for any convex $n$-gon with the semi-circle property, one can find all of the nearest neighbors among its vertices one for each vertex in total time $O(n)$.

Theorem 19.1. Given a convex polygon $P = p_1, \ldots, p_n$ with semi-circle property on the edge $p_np_0$, we have that $\forall i, 0 < i < n$, $p_{i-1}p_i < p_{i-1}p_{i+1}$ and $p_i p_{i+1} < p_{i-1} p_{i+1}$.

Proof. Any angle inscribed on the semi-circumference is $\frac{\pi}{2}$; thus $\measuredangle p_0 p_i p_n \geq \frac{\pi}{2}$, and this implies that $p_{i-1} p_i p_{i+1} \geq \frac{\pi}{2}$ as it contains the previous angle for convexity. If we consider now the triangle $\triangle p_{i-1} p_i p_{i+1}$ we know than that $p_{i-1} p_{i+1}$ is the bigger side, being opposed to an obtuse angle, thus the theorem.

This gives us the following theorem:

Theorem 19.2. Given a convex polygon $P = p_1, \ldots, p_n$ with semi-circle property on the edge $p_np_0$, we have that $\forall i, 0 \leq i \leq n$, the nearest neighbor in $P$ of $p_i$ is either $p_{i-1}$ or $p_{i+1}$.
Proof. Consider the triangle $\triangle p_{i-2}p_{i-1}p_i$: by theorem 19.1 we know that $p_ip_{i-1} < p_ip_{i-2}$; now, if we remove the triangle from the polygon, that is, if we consider $\mathcal{P} \setminus \{p_i\}$ the polygon still has the semi-circle property on $p_ip_j$, and we can apply again theorem 19.1 to obtain that $p_ip_{i-1} < p_ip_{i-2} < p_ip_{i-3}$. Iterating the procedure we have that $\forall 0 \leq j < i - 1, p_ip_{i-1} < p_ip_j$. The reasoning in the other direction is symmetrical and we have that $\forall i + 1 < j \leq n, p_ip_{i+1} < p_ip_j$. □

This gives us a straightforward algorithm for computing the nearest neighbors of the vertices of $\mathcal{P}$ in linear time, as we just have to find for each vertex the shorter of the two incident edges.

**Assignment 20** Consider a set $S$ of $n$ points in $\mathbb{R}^2$ and suppose that all inter-point distances are distinct. A point $q \in S$ is an east-northeast neighbor of $p \in S$, with $p \neq q$, if:

1. the $y$ coordinate of $q$ is greater than or equal to the $y$-coordinate of $p$ and
2. the edge $(p, q)$ has a slope between 0 and 1.

Define a graph $\mathcal{G}$ with vertex set $S$ by adding an edge between each point and its nearest east-northeast neighbor. Now add edges to $\mathcal{G}$ between each point and its nearest neighbor to the north-northeast, north-northwest, west-northwest, etc. (for all eight possibilities). Prove that $\mathcal{G}$ contains all edges of the EMST of $S$.

Proof. (By absurd) Consider an edge $pq$ of the EMST $T$ of $S$ that, without loss of generality, goes in ENE direction. Suppose that it does not belong to the 8-quadrant nearest neighbor graph: then $q$ is not an ENE nearest neighbor of $p$, nor $p$ is a WSW nearest neighbor of $q$. Then, there exist points $q'$ and $p'$ that are the respectively the ENE NN of $p$ and the WSW NN of $q$. If we split the tree $T$ removing the edge $pq$, we obtain two trees $T_p$, where $p$ lies, and $T_q$, where $q$ lies.
Now, consider the position of $q'$ and $p'$; we have three cases:

$q' \in T_q$: If we consider the tree $T_p \cup \overrightarrow{pq} \cup T_q$, it has a total length smaller than the original $T$, thus contradicting that $T$ is minimum.

$p' \in T_p$: If we consider the tree $T_p \cup \overrightarrow{ pq } \cup T_q$, it has again a total length smaller than the original $T$.

$q' \in T_p$ and $p' \in T_q$: We notice that the circular sector centered in $p$, with ray $\overrightarrow{pq}$ and lying in the ENE quadrant, is entirely contained in the circle of ray $\overrightarrow{pq}$ centered in $q$; then the tree $T_p \cup \overrightarrow{pq} \cup T_q$ has length smaller than $T$.

\[ \square \]

**Assignment 21** A simple polygon is called orthogonal if all its edges are horizontal or vertical (of course, this is a non-exclusive “or”!) Devise an algorithm that runs in $O(n)$ time on $n$-vertex orthogonal polygons to check whether the polygon has a non-empty kernel and, if it is indeed not empty, to compute the kernel (please note that it does not suffice to quote the linear-time algorithm for computing the kernel of a general simple polygon. As a matter of fact, computing the kernel is considerably simpler for orthogonal polygons!).

This exercise is in O’Rourke’s book (O’Rourke, 1998, Ex. 3.9.2.5, p.97). The algorithm for computing the kernel of a simple polygon in linear time is due to Lee and Preparata (Preparata and Shamos, 1985, Sec. 7.2.6, pp.299-306).

We know that the kernel of a polygon is the intersection of its left half-planes (assuming the polygon is in counter-clockwise order): we can then easily come up with an algorithm intersecting orthogonal half-planes in $O(n)$ time.

The following algorithm returns the kernel of an orthogonal (isothonic) polygon; when $x_{\text{min}} > x_{\text{max}}$ or $y_{\text{min}} > y_{\text{max}}$ the kernel is empty.
**INTERSECTORTHOGONALHALFPLANES**(P)

1. \(x_{\text{min}} \leftarrow -\infty\)
2. \(x_{\text{max}} \leftarrow \infty\)
3. \(y_{\text{min}} \leftarrow -\infty\)
4. \(y_{\text{max}} \leftarrow \infty\)
5. \(\text{for } i \leftarrow 1 \text{ to length}[P]\)
6. \(\text{if } P[i].x = P[i + 1].x\)
7. \(\text{then if } P[i].y < P[i + 1].y\)
8. \(\quad x_{\text{max}} \leftarrow \min(P[i].x, x_{\text{max}})\)
9. \(\quad \text{else } x_{\text{min}} \leftarrow \max(P[i].x, x_{\text{min}})\)
10. \(\text{else if } P[i].x < P[i + 1].x\)
11. \(\quad y_{\text{min}} \leftarrow \max(P[i].y, y_{\text{min}})\)
12. \(\quad \text{else } y_{\text{max}} \leftarrow \min(P[i].y, y_{\text{max}})\)
13. \(\text{return } [x_{\text{min}}, y_{\text{min}}] \times [x_{\text{max}}, y_{\text{max}}]\)
Assignment 22 Recall that an approximate TSP tour for a set $S$ of $n$ points in the Euclidean plane can be obtained by computing $EMST(S)$, picking one point of $S$ (and, thus, one node of $EMST(S)$) as a root, and applying a standard tree traversal e.g., an in-order traversal to $EMST(S)$, where nodes already visited are bypassed. Theory tells us that the approximation factor achieved by this algorithm is no worse than 2. Show that there exist configurations of $n$ points for which the approximation factor will indeed approach 2 as $n$ grows, no matter which node of $EMST(S)$ is picked as the root. (No formal proof is needed, but your point configurations should be simple enough to admit a reasonably “clean” argument.)

Given an $n$, with $n$ even, we put $\frac{n}{2}$ points equally distributed over a circumference of ray $r$. We then build a concentric circle with an arbitrary small ray difference $\epsilon$, so that its ray is $r + \epsilon$. We put the remaining points on the new circle in a way that each one is as closer as possible to one of the previous points. This figure shows the EMST of a set of points so obtained for $n = 16$, the 2-Approximate TSP by tree traversal heuristic, and the optimal TSP:

It is intuitive (also if a bit complicated to put formally down) that the total length of the approximation tends to $APX \rightarrow 4\pi r$, while the optimal TSP length tends to $OPT \rightarrow 2\pi r$
Assignment 23 Consider an $n$-vertex polygon $P = (p_0, p_1, p_2, \ldots, p_{n-1})$. If two edges $(p_i, p_{i+1})$ and $(p_j, p_{j+1})$ intersect then we replace them by (1) a new edge between $p_i$ and $p_j$ and (2) a new edge between $p_{i+1}$ and $p_{j+1}$. (All vertex indices are taken modulo $n$.) Such a replacement operation is called a “smoothing” operation. Then we re-number the vertices of the resulting polygon, thus obtaining a new polygon $P' = (p'_0, p'_1, p'_2, \ldots, p'_{n-1})$ (of course, $\{p_0, p_1, p_2, \ldots, p_{n-1}\} = \{p'_0, p'_1, p'_2, \ldots, p'_{n-1}\}$.) Starting with an $n$-vertex polygon $P$, we will randomly pick pairs of intersecting edges and apply smoothing operations until we arrive at a polygon which is simple. Is it always guaranteed that this process stops? I.e., is it guaranteed that we will eventually arrive at a polygon which is simple?

It is guaranteed that the process stops.

Proof. We will show that the process stops by showing that the smoothing operation (2-opt) strictly decreases the perimeter of the polygon. As each operation strictly decreases the perimeter, and the number of possible polygons through a set of points is bounded (it is $\frac{(n-1)!}{2}$, where $n$ is the number of points), the process has to stop. Note that there also the perimeter of a polygon through a given set of points is bounded (it is the length of the traveling salesman tour), and that this process does not always produce the polygon with the smallest perimeter (well, that would be nice because then we could solve TSP in $O(n^3)$). Note also that the operation maintains the polygon connected. Consider two crossing edges $(p_i, p_{i+1})$ and $(p_j, p_{j+1})$, and call $q$ the point where they cross:

Then, using triangle inequalities we have that:

\[
\begin{align*}
|pq| &< |pq| + |qj| \\
|pi+1pj+1| &< |pi+1q| + |qj+1| \\
|pj| + |pi+1pj+1| &< |pi+1| + |pj| + |j+1|
\end{align*}
\]

Where the last term is obtained by summing the previous two inequalities. Then, as every smoothing operation substitutes the two crossing edges (terms on the right of the inequality) with the edges between their starting/ending points (terms on the left), the total perimeter is strictly decreasing. \qed
9 Additional material

Theorem 23.1 (Existence of a strictly convex vertex). Every simple polygon $\mathcal{P} = \{p_1, \ldots, p_n\}$ has at least a strictly convex vertex.

Proof. Let us consider the vertex $p_i \in \mathcal{P}$ with the smallest $y$ coordinate: if several vertices have this property, we take the one with the bigger $x$ coordinate (we consider in practice the smallest vertex in the lexicographical ordering $(y, -x)$). If we consider the $x$-aligned line passing through this point, all the remaining points in the polygon must lie above this line, including $p_{i-1}$ and the vertex $p_{i+1}$ must be strictly above this line. Then $p_i$ is strictly convex. \hfill \Box

Theorem 23.2 (Existence of a diagonal). Every simple polygon $\mathcal{P} = \{p_1, \ldots, p_n\}$ with $n > 3$ has at least one diagonal.

Proof. From theorem 23.1 we know that $\mathcal{P}$ has a strictly convex vertex $p_i$. We consider now the segment $p_{i-1}p_{i+1}$: if it is contained in $\mathcal{P}$ it is a diagonal and we have concluded. If it is not, then the triangle $\Delta p_{i-1}p_ip_{i+1}$ must contain at least another point of $\mathcal{P}$: we move then a line $L$ parallel to $p_{i-1}p_{i+1}$ from $p_i$ to $p_{i-1}p_{i+1}$ until we meet a vertex that we will call $x$. Now, if we consider the area given by the half-plane determined by $L$ containing $p_i$ intersected the triangle $\Delta p_{i-1}p_ip_{i+1}$, we notice that it is entirely contained in the polygon and thus $p_{i}x$ is a diagonal. \hfill \Box

Theorem 23.3 (Existence of a $n - 2$ triangulation). Every simple polygon $\mathcal{P} = \{p_1, \ldots, p_n\}$ with $n \geq 3$ can be triangulated with $n - 2$ triangles.
\textbf{Proof.} (By induction)

\textbf{(base case)} If $n = 3$, then the polygon is a triangle and has then a trivial triangulation of one triangle.

\textbf{(inductive step)} If $n > 3$, we know by theorem 23.2 that the polygon has a diagonal that splits it in two pieces. If one piece has $m$ vertices, than the other one has $n - m + 2$ vertices. Using the inductive hypothesis, they have a triangulation of, respectively, $m - 2$ and $n - m$ triangles. A triangulation of the whole polygon can then be obtained merging these two and obtaining one of $m - 2 + n - m = n - 2$ triangles. \hfill \square

\textbf{Theorem 23.4.} The number of possible polygons through a set of $n$ points is $\frac{(n - 1)!}{2n}$.

\textbf{Proof.} The number is given by the number of possible permutations of the set of points, that is $n!$. But we have to notice that this way we count a polygon several times, only that is has different “starting points”. More precisely, each polygon is counted $n$ times (for the $n$ different starting points) in one direction and $n$ times in the opposite direction. This gives that the total number of polygons through a set of $n$ points is:

$$\frac{n!}{2n} = \frac{(n - 1)!}{2}$$

\hfill \square
References


