Segmented Least Squares

Abhiram Ranade

February 29, 2016

Dynamic programming for optimization problems

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.
4. For each way, determine the best objective value that can be acheived, given that the remaining decisions are taken in the best manner.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.
4. For each way, determine the best objective value that can be acheived, given that the remaining decisions are taken in the best manner.
5. Return the way that gives the minimum amongst all ways.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.
4. For each way, determine the best objective value that can be acheived, given that the remaining decisions are taken in the best manner.
5. Return the way that gives the minimum amongst all ways.

Sometimes step 4 can be implemented by recursing on smaller instances.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.
4. For each way, determine the best objective value that can be acheived, given that the remaining decisions are taken in the best manner.
5. Return the way that gives the minimum amongst all ways.

Sometimes step 4 can be implemented by recursing on smaller instances.
Sometimes the recursion tree will contain calls to solve the same subinstance in different parts.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.
4. For each way, determine the best objective value that can be acheived, given that the remaining decisions are taken in the best manner.
5. Return the way that gives the minimum amongst all ways.

Sometimes step 4 can be implemented by recursing on smaller instances.
Sometimes the recursion tree will contain calls to solve the same subinstance in different parts.

Key dynamic programming idea: Solve each subinstance only once, rememer it. Reuse the value if needed later.

Dynamic programming for optimization problems

1. Pretend you will do a brute force search.
2. Decide an order in which to take the decisions.
3. Consider all ways of taking the first decision.
4. For each way, determine the best objective value that can be acheived, given that the remaining decisions are taken in the best manner.
5. Return the way that gives the minimum amongst all ways.

Sometimes step 4 can be implemented by recursing on smaller instances.
Sometimes the recursion tree will contain calls to solve the same subinstance in different parts.

Key dynamic programming idea: Solve each subinstance only once, rememer it. Reuse the value if needed later.

How does this compare to greedy?

Dynamic programming recap continued

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

As in computing Fibonacci numbers using $F(n) = F(n-1) + F(n-2)$.

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

As in computing Fibonacci numbers using $F(n) = F(n-1) + F(n-2)$.

Key requirement: Recurrence relating the cost of given instance to the costs of subinstances.

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

As in computing Fibonacci numbers using $F(n) = F(n-1) + F(n-2)$.

Key requirement: Recurrence relating the cost of given instance to the costs of subinstances.

Note that usually we dont just need the cost of the best solution but the solution itself. However, typically if we get a way to find the best cost, we can get the best solution with mild additional effort.

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

As in computing Fibonacci numbers using $F(n) = F(n-1) + F(n-2)$.

Key requirement: Recurrence relating the cost of given instance to the costs of subinstances.

Note that usually we dont just need the cost of the best solution but the solution itself. However, typically if we get a way to find the best cost, we can get the best solution with mild additional effort.

Example:

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

As in computing Fibonacci numbers using $F(n) = F(n-1) + F(n-2)$.

Key requirement: Recurrence relating the cost of given instance to the costs of subinstances.

Note that usually we dont just need the cost of the best solution but the solution itself. However, typically if we get a way to find the best cost, we can get the best solution with mild additional effort.

Example:
Solution = minimum weight subset of non-overlapping intervals.

Dynamic programming recap continued

Once you know what subinstances are to be solved, do not use recursion at all, but build up the solution bottom

As in computing Fibonacci numbers using $F(n) = F(n-1) + F(n-2)$.

Key requirement: Recurrence relating the cost of given instance to the costs of subinstances.

Note that usually we dont just need the cost of the best solution but the solution itself. However, typically if we get a way to find the best cost, we can get the best solution with mild additional effort.

Example:
Solution = minimum weight subset of non-overlapping intervals.
Cost = weight of the minimum weight subset.

Segmented least squares

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.

Assumption: The user intended to draw a sequence of an unknown number of connected line segments.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.

Assumption: The user intended to draw a sequence of an unknown number of connected line segments.

Output: The piecewise linear curve that the user intended to draw.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.

Assumption: The user intended to draw a sequence of an unknown number of connected line segments.

Output: The piecewise linear curve that the user intended to draw.

The user will not be a perfect artist, so there will be some error. The points need not be on the inferred curve.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.

Assumption: The user intended to draw a sequence of an unknown number of connected line segments.

Output: The piecewise linear curve that the user intended to draw.

The user will not be a perfect artist, so there will be some error. The points need not be on the inferred curve.

However, the deviation should be small.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.
Assumption: The user intended to draw a sequence of an unknown number of connected line segments.
Output: The piecewise linear curve that the user intended to draw.

The user will not be a perfect artist, so there will be some error. The points need not be on the inferred curve.

However, the deviation should be small.

"Degenerate solution:" connect points by straight line segments.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.
Assumption: The user intended to draw a sequence of an unknown number of connected line segments.
Output: The piecewise linear curve that the user intended to draw.

The user will not be a perfect artist, so there will be some error. The points need not be on the inferred curve.

However, the deviation should be small.

"Degenerate solution:" connect points by straight line segments.

Deviation $= 0$, but unlikely to be the right solution.

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.
Assumption: The user intended to draw a sequence of an unknown number of connected line segments.
Output: The piecewise linear curve that the user intended to draw.

The user will not be a perfect artist, so there will be some error. The points need not be on the inferred curve.

However, the deviation should be small.

"Degenerate solution:" connect points by straight line segments.

Deviation $= 0$, but unlikely to be the right solution.

Usually: number of pieces $\ll$ number of pieces

Segmented least squares

Input: Sequence of points $p_1, \ldots, p_n$ in the plane, say generated by mouse drag in order.
Assumption: The user intended to draw a sequence of an unknown number of connected line segments.
Output: The piecewise linear curve that the user intended to draw.

The user will not be a perfect artist, so there will be some error. The points need not be on the inferred curve.

However, the deviation should be small.

"Degenerate solution:" connect points by straight line segments.

Deviation $= 0$, but unlikely to be the right solution.

Usually: number of pieces $\ll$ number of pieces

How do we balance the goals of having a small number of pieces and yet acheiving a close fit?

Warmup: Fitting the best single segment

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

$$LSE(P) = \min_{a,b} \sum_i (y_i - ax_i - b)^2$$

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

$$LSE(P) = \min_{a,b} \sum_i (y_i - ax_i - b)^2$$

Solution: Set derivatives w.r.t. a, b to 0 and solve, giving

$$a = \frac{n\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a\sum_i x_i}{n}$$

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

$$LSE(P) = \min_{a,b} \sum_i (y_i - ax_i - b)^2$$

Solution: Set derivatives w.r.t. a, b to 0 and solve, giving

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

$LSE(P) = \sum_i (y_i - ax_i - b)^2$ with a, b as calculated above.

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

$$LSE(P) = \min_{a,b} \sum_i (y_i - ax_i - b)^2$$

Solution: Set derivatives w.r.t. a, b to 0 and solve, giving

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

$LSE(P) = \sum_i (y_i - ax_i - b)^2$ with a, b as calculated above.
Time to calculate $LSE(p_1, \ldots, p_n) = O(n)$.

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

$$LSE(P) = \min_{a,b} \sum_i (y_i - ax_i - b)^2$$

Solution: Set derivatives w.r.t. a, b to 0 and solve, giving

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

$LSE(P) = \sum_i (y_i - ax_i - b)^2$ with a, b as calculated above.
Time to calculate $LSE(p_1, \ldots, p_n) = O(n)$.

Exercise: Find $LSE(\{(0,0),(10,1),(20,0)\})$

Warmup: Fitting the best single segment

Find a, b s.t. line $y = ax + b$ best fits the sequence of points $P = (x_1, y_1), \ldots, (x_n, y_n)$.

Best fit: minimize square of vertical distance of each point from the line.

"Least square error"

$$LSE(P) = \min_{a,b} \sum_i (y_i - ax_i - b)^2$$

Solution: Set derivatives w.r.t. a, b to 0 and solve, giving

$$a = \frac{n\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a\sum_i x_i}{n}$$

$LSE(P) = \sum_i (y_i - ax_i - b)^2$ with a, b as calculated above.
Time to calculate $LSE(p_1, \ldots, p_n) = O(n)$.

Exercise: Find $LSE(\{(0,0), (10,1), (20,0)\})$ $\quad 2/3,\ a = 0, b = 1/3$

Segmented least squares: formal definition

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum $Cost$, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum $Cost$, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum $Cost$, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

- $c = 0 \Rightarrow L =$

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum *Cost*, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

- $c = 0 \Rightarrow L = n/2$

segments must be disjoint

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum $Cost$, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

- $c = 0 \Rightarrow L = n/2$ segments must be disjoint
- $c = \infty \Rightarrow L =$

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum *Cost*, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

- $c = 0 \Rightarrow L = n/2$ segments must be disjoint
- $c = \infty \Rightarrow L = 1$

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum *Cost*, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

- $c = 0 \Rightarrow L = n/2$ segments must be disjoint
- $c = \infty \Rightarrow L = 1$

Hope: An intermediate value will give us a good partition.

Segmented least squares: formal definition

Input: Points $P = \{p_1, \ldots, p_n\}$. $p_i = (x_i, y_i)$. $x_1 < x_2 < \ldots < x_n$.
Output: Number L of segments, their starting points $s_2, \ldots, s_L$ of minimum *Cost*, where

$$Cost(L, s_2, \ldots, s_L) = cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$$

where c is a suitable constant, and $s_1 = 1, s_{L+1} = n + 1$.

How to choose c:

- $c = 0 \Rightarrow L = n/2$ — segments must be disjoint
- $c = \infty \Rightarrow L = 1$

Hope: An intermediate value will give us a good partition.
Note: c to be fixed first; not decided by the algorithm.

Brute force solution

Brute force solution

We need to decide $L, s_2, \ldots, s_L$

Brute force solution

We need to decide $L, s_2, \ldots, s_L$

Try all possible choices for each in some order.

Brute force solution

We need to decide $L, s_2, \ldots, s_L$

Try all possible choices for each in some order.

Strategy: Start with all possible choices $s = 1, \ldots, n-1$ for s_L.

Brute force solution

We need to decide $L, s_2, \ldots, s_L$

Try all possible choices for each in some order.

Strategy: Start with all possible choices $s = 1, \ldots, n-1$ for s_L.

Cost(Optimal partitioning of $p_1, \ldots, p_n$) =
$\min_s\{$Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)$\}$

Brute force solution

We need to decide $L, s_2, \ldots, s_L$

Try all possible choices for each in some order.

Strategy: Start with all possible choices $s = 1, \ldots, n-1$ for s_L.

Cost(Optimal partitioning of $p_1, \ldots, p_n$) =
$\min_s$ {Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)}

How do we find the cost of an optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at p_s?

Brute force solution

We need to decide $L, s_2, \ldots, s_L$

Try all possible choices for each in some order.

Strategy: Start with all possible choices $s = 1, \ldots, n-1$ for s_L.

Cost(Optimal partitioning of $p_1, \ldots, p_n$) =
$\min_s$ {Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)}

How do we find the cost of an optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at p_s?

As usual, we will concentrate on finding the optimal cost; the actual partitioning can then be found out easily.

Lemma

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $s_L = s$

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\qquad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\quad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$
First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\quad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$
First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.
$\geq$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+LSE(p_s, \ldots, p_n) + c$

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) $=$
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\qquad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$
First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.
$\geq$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+LSE(p_s, \ldots, p_n) + c$
But this is the cost of some partitioning of

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) =
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\qquad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$
First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.
$\geq$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+LSE(p_s, \ldots, p_n) + c$
But this is the cost of some partitioning of $p_1, \ldots, p_n$ with last segment beginning at p_s.

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) =
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)

$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\quad s_L = s$

$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$

First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.

$\geq$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+LSE(p_s, \ldots, p_n) + c$

But this is the cost of some partitioning of $p_1, \ldots, p_n$ with last segment beginning at p_s.

$\geq$ Cost(Optimal partitioning s.t. last segment begins at p_s)

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) =
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\quad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$
First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.
$\geq$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+LSE(p_s, \ldots, p_n) + c$
But this is the cost of some partitioning of $p_1, \ldots, p_n$ with last segment beginning at p_s.
$\geq$ Cost(Optimal partitioning s.t. last segment begins at p_s)

Thus we must have equality through out. $\square$

Lemma

Cost(Optimal partitioning s.t. last segment begins at p_s) =
Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Proof: Suppose optimal partitioning of $p_1, \ldots, p_n$ with last segment starting at s has L segments.

Cost(Optimal partitioning s.t. last segment begins at p_s)
$= cL + \sum_{j=1}^{L} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1})$ $\qquad s_L = s$
$= c(L-1) + \sum_{j=1}^{L-1} LSE(p_{s_j}, \ldots, p_{s_{j+1}-1}) + LSE(p_s, \ldots, p_n) + c$
First 2 terms give the cost of some partitioning of $p_1, \ldots, p_{s-1}$.
$\geq$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+LSE(p_s, \ldots, p_n) + c$
But this is the cost of some partitioning of $p_1, \ldots, p_n$ with last segment beginning at p_s.
$\geq$ Cost(Optimal partitioning s.t. last segment begins at p_s)

Thus we must have equality through out. $\square$

Note that optimal partitioning of $p_1, \ldots, p_{s-1}$ need not have $L-1$ segments.

Implication

Implication

Lemma:

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) $=$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) = Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) = $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) = Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) = $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i)$ = Cost of optimal partitioning of $p_1, \ldots, p_i$

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) = Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) = $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i)$ = Cost of optimal partitioning of $p_1, \ldots, p_i$ $\qquad C(0) = 0$

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) = Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) = $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i) =$ Cost of optimal partitioning of $p_1, \ldots, p_i$ $\qquad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) = Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) = $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i)$ = Cost of optimal partitioning of $p_1, \ldots, p_i$ $\qquad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

In general: $C(i) = \min_{s=1..i-1} C(s-1) + c + LSE(p_s, \ldots, p_i)$

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) $=$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) $=$ $\min\limits_{s}$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i) =$ Cost of optimal partitioning of $p_1, \ldots, p_i$ $\quad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

In general: $C(i) = \min\limits_{s=1..i-1} C(s-1) + c + LSE(p_s, \ldots, p_i)$

So we make a table T[0..n], and store $C(i)$ in $T[i]$.

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) $=$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) $=$ $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i) =$ Cost of optimal partitioning of $p_1, \ldots, p_i$ $\qquad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

In general: $C(i) = \min_{s=1..i-1} C(s-1) + c + LSE(p_s, \ldots, p_i)$

So we make a table T[0..n], and store $C(i)$ in $T[i]$.
Convenient to have $T[0] = 0$. $T[n]$ will store final answer.

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) $=$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) $=$ $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i) =$ Cost of optimal partitioning of $p_1, \ldots, p_i$ $\quad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

In general: $C(i) = \min_{s=1..i-1} C(s-1) + c + LSE(p_s, \ldots, p_i)$

So we make a table T[0..n], and store $C(i)$ in $T[i]$.
Convenient to have $T[0] = 0$. $T[n]$ will store final answer.

Fill the table in increasing order of i.

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) $=$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) $=$ $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i) =$ Cost of optimal partitioning of $p_1, \ldots, p_i$ $\quad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

In general: $C(i) = \min_{s=1..i-1} C(s-1) + c + LSE(p_s, \ldots, p_i)$

So we make a table T[0..n], and store $C(i)$ in $T[i]$.
Convenient to have $T[0] = 0$. $T[n]$ will store final answer.

Fill the table in increasing order of i.
Filling ith entry: $O(n^2)$ time, $O(n)$ if $LSE(p_s, \ldots, p)i)$ known.

Implication

Lemma: Cost(Optimal partitioning s.t. last segment is from p_s) $=$ Cost(Optimal partitioning of $p_1, \ldots, p_{s-1}$) $+c + LSE(p_s, \ldots, p_n)$

We also know that Cost(Optimal partitioning of $p_1, \ldots, p_n$) $=$ $\min_s$ Cost(optimal partitioning of $p_1, \ldots, p_n$, last segment from p_s)

$C(i) =$ Cost of optimal partitioning of $p_1, \ldots, p_i$ $\quad\quad C(0) = 0$

$$C(n) = \min_{s=1..n-1} C(s-1) + c + LSE(p_s, \ldots, p_n)$$

In general: $C(i) = \min_{s=1..i-1} C(s-1) + c + LSE(p_s, \ldots, p_i)$

So we make a table T[0..n], and store $C(i)$ in $T[i]$.
Convenient to have $T[0] = 0$. $T[n]$ will store final answer.

Fill the table in increasing order of i.
Filling ith entry: $O(n^2)$ time, $O(n)$ if $LSE(p_s, \ldots, p)i)$ known.
Total: $O(n^3)$. By precomputing $LSE(p_s, \ldots, p_i)$: $O(n^2)$

The program

```
T[0] = 0;

For i=1 to n
  For k=1 to i
    T[i] = min(T[i], T[k-1] + c + LSE(k,i))
  end
end
```

LSE(k,i) : use formula, $O(n)$ time.

The program

```
T[0] = 0;

For i=1 to n
  For k=1 to i
    T[i] = min(T[i], T[k-1] + c + LSE(k,i)
  end
end
```

LSE(k,i) : use formula, $O(n)$ time.

Optimization:

$$\sum_{j=i}^{j=k} x_i = \sum_{i=1}^{i=k} x_i - \sum_{i=1}^{i=j-1} x_i$$

The program

```
T[0] = 0;

For i=1 to n
  For k=1 to i
    T[i] = min(T[i], T[k-1] + c + LSE(k,i))
  end
end
```

LSE(k,i) : use formula, $O(n)$ time.

Optimization:

$$\sum_{j=i}^{j=k} x_i = \sum_{i=1}^{i=k} x_i - \sum_{i=1}^{i=j-1} x_i$$

Precompute $\sum_{i=1}^{i=j} x_i$ for all j.

The program

```
T[0] = 0;

For i=1 to n
  For k=1 to i
    T[i] = min(T[i], T[k-1] + c + LSE(k,i))
  end
end
```

LSE(k,i) : use formula, $O(n)$ time.

Optimization:

$$\sum_{j=i}^{j=k} x_i = \sum_{i=1}^{i=k} x_i - \sum_{i=1}^{i=j-1} x_i$$

Precompute $\sum_{i=1}^{i=j} x_i$ for all j.

All terms involved in $LSE(i,j)$ are of the type above. So after precomputation each LSE can be calculated in $O(1)$ time.

The program

```
T[0] = 0;

For i=1 to n
  For k=1 to i
    T[i] = min(T[i], T[k-1] + c + LSE(k,i))
  end
end
```

LSE(k,i) : use formula, $O(n)$ time.

Optimization:

$$\sum_{j=i}^{j=k} x_i = \sum_{i=1}^{i=k} x_i - \sum_{i=1}^{i=j-1} x_i$$

Precompute $\sum_{i=1}^{i=j} x_i$ for all j.

All terms involved in $LSE(i,j)$ are of the type above. So after precomputation each LSE can be calculated in $O(1)$ time.
So total $O(n^2)$.

Remarks

Remarks

The number of segments L and the starting points $s_2, \ldots, s_L$ can be found by examining the relationships between the table entries.

Remarks

The number of segments L and the starting points $s_2, \ldots, s_L$ can be found by examining the relationships between the table entries.

Note that the main Lemma relates the cost of any solution with last segment starting at s to the cost of the solution for the first s points.

Remarks

The number of segments L and the starting points $s_2, \ldots, s_L$ can be found by examining the relationships between the table entries.

Note that the main Lemma relates the cost of any solution with last segment starting at s to the cost of the solution for the first s points.

The entire solution contains the solution for the first s points.

Remarks

The number of segments L and the starting points $s_2, \ldots, s_L$ can be found by examining the relationships between the table entries.

Note that the main Lemma relates the cost of any solution with last segment starting at s to the cost of the solution for the first s points.

The entire solution contains the solution for the first s points.

Hence "optimal substructure".

Remarks

The number of segments L and the starting points $s_2, \ldots, s_L$ can be found by examining the relationships between the table entries.

Note that the main Lemma relates the cost of any solution with last segment starting at s to the cost of the solution for the first s points.

The entire solution contains the solution for the first s points.

Hence "optimal substructure".

Note that the Lemma cannot be proved if cost has cL^2 instead of cL.

Remarks

The number of segments L and the starting points $s_2, \ldots, s_L$ can be found by examining the relationships between the table entries.

Note that the main Lemma relates the cost of any solution with last segment starting at s to the cost of the solution for the first s points.

The entire solution contains the solution for the first s points.

Hence "optimal substructure".

Note that the Lemma cannot be proved if cost has cL^2 instead of cL.

But even for this cost function you will be able to find the optimal solution in polytime via a different dynamic programming strategy.