COMP 3711 – Honors Design and Analysis of Algorithms
2021 Spring Semester – Written Assignment # 3
Double-Sized Assignment (worth twice the other assignments)
Distributed March 22, 2021 – Typos Fixed March 27, 2021
~~Due April 7, 2021~~ Due April 10, 2021

Your solutions should contain (i) your name, (ii) your student ID #, and (iii) your email address

<u>Some Notes:</u>

- Please write clearly and briefly. Your solutions should follow the guidelines given at
  https://canvas.ust.hk/courses/36180/pages/assignment-submission-guidelines

  In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.

- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page.
  You must acknowledge individuals who assisted you, or sources where you found solutions. Failure to do so will be considered plagiarism.

- The term *Documented Pseudocode* means that you must include clear comments *inside* your pseudocode.

- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.

- Please make a *copy* of your assignment before submitting it. If we can't find your submission, we will ask you to resubmit the copy.

- Submit a SOFTCOPY of your assignment to CASS by the deadline. The softcopy should be one PDF file (no word or jpegs permitted, nor multiple files).

  If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

- Problems 4, 5, 6 and 9 ask you to *give a (Dynamic Programming) recurrence* This means explicitly writing the mathematical recurrence describing the DP in the way that we have done many times in class.

- **Typos Fixed:** P4. Diagram and comments in Rules, Recommendation and Hints changed. (Previous content illustrated a different type of BST.)

P1: [8 pts] **Huffman Coding**

The table below lists a vocabulary of 15 characters (a to o) and their frequencies in a document. Apply Huffman coding (following the algorithm on Page 11 of the 10_Huffman.pptx lecture notes) to construct an optimal codebook. Your solution should contain three parts:

(a) A full Huffman tree.

(b) The final codebook that contains the binary codewords representing a to o (sorted in the alphabetical order on a to o).

(c) The codebook from part (b) but now sorted by the increasing lengths of the codewords.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a_i	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
$f(a_i)$	36	6	9	10	58	7	17	29	45	2	3	12	15	26	28

Rules, Recommendations and Hints:

- In part (a) you should explicitly label each edge in the tree with a '0' or '1'. You should also label each leaf with its corresponding character a_i and $f(a_i)$ value.
- Part (b) should be a table with 2 columns. The first column should be the letters a to o. The second column should be the codeword associated with each character.
- In part (c) you should break ties using the alphabetical order of the characters. For example, if a, d and e all have codewords of length 5, then write the codeword for a on top of the codeword for d on top of the codeword for e.

P2: [12 pts] **Optimum Binary Search Trees**

You are given an alphabet $\mathcal{A}$ containing 8 characters (in alphabetical order) and their associated frequencies in the table below. Create an optimal binary search tree of $\mathcal{A}$ to minimize the expected time required to search for a character in this alphabet.

i	1	2	3	4	5	6	7	8
a_i	A	B	C	D	E	F	G	H
$f(a_i)$	5	8	1	6	4	7	3	2

To solve this problem, you need to apply the Dynamic Programming algorithm shown in Page 19 of 13_DP_interval lecture note.

(a) Present your result by showing the final $e[i,j]$ and $root[i,j]$ tables. (The same as the two tables shown in the 13_BST_Worked_Example lecture notes).

(b) What is the minimum search time cost of a BST for $\mathcal{A}$?

(c) Execute the "Construct-BST" algorithm on Page 20 of 13_DP_interval lecture note. Write down your function calls and draw the final optimal Binary Search Tree result (the corresponding legal solution).

P3: [15 pts] **Chain Matrix Multiplication**

(a) Recall that the cost of multiplying matrices together is the total number of scalar multiplications used.

Let $\text{cost}((A_1A_2)A_3)$ be the cost of first multiplying A_1A_2 and then multiplying their result by A_3; Let $\text{cost}(A_1(A_2A_3))$ be the cost of first multiplying A_2A_3 and then multiplying their result by A_1. In class we saw an example in which

$$\frac{\text{cost}(A_1(A_2A_3))}{\text{cost}((A_1A_2)A_3)} = 10$$

Multiplying in the "correct" order could reduce the total cost by 90%.

For this problem, for every integer $t > 1$, define a quadruple $\left(p_0^{(t)}, p_1^{(t)}, p_2^{(t)}, p_3^{(t)}\right)$ such that, if A_i has dimensions $p_{i-1}^{(t)} \times p_i^{(t)}$ then

$$\frac{\text{cost}(A_1(A_2A_3))}{\text{cost}((A_1A_2)A_3)} \geq t. \qquad (1)$$

This shows that the gain of using the "correct" order is unbounded.

Your solution should be in two parts

(i) Give the formulas for $\left(p_0^{(t)}, p_1^{(t)}, p_2^{(t)}, p_3^{(t)}\right)$

(ii) Prove the correctness of Equation (1) for your given values.

(b) Given a chain of 7 matrices $< A_1, A_2, ..., A_7 >$ and their dimensions (as provided in the table below) fully parenthesize the product $A_1A_2...A_7$ in a way that minimizes the number of scalar multiplications required to multiply them all together.

Matrix	A_1	A_2	A_3	A_4	A_5	A_6	A_7
Dimension	6 x 7	7 x 3	3 x 1	1 x 2	2 x 4	4 x 5	5 x 3

To solve this problem, you need to apply the Dynamic Programming algorithm shown in Page 9 of 13b_Matrix_Multiplication lecture note.

(i) Present your result by showing the final $m[i, j]$ and $s[i, j]$ tables. Present them using the same structure as the two tables shown on Page 10 of 13b_Matrix_Multiplication lecture notes.

(ii) What is the minimum number of scalar multiplications needed to compute the product $A_1A_2...A_7$?

(iii) Present the optimal parenthesized sequence for multiplying $A_1A_2...A_7$ (following the formatting of the sequence shown on Page 8 of 13c_Matrix_Mutiplication_Worked_Example.pptx lecture notes).

P4: [25 pts] **More Optimum Binary Search Trees**

In class we developed a $O(n^3)$ time solution for the optimal Binary Search tree problem.

In this problem you need to modify that solution to construct an optimal tree of given restricted height. The height of a tree T is

$$height(T) = \max_{1 \le t \le n} d(a_t)$$

where $d(a_t)$ is the depth of the node corresponding to a_t in tree T.

The optimal tree constructed by the algorithm could have height as large as $n-1$. One standard constraint that occurs in practice is to restrict the tree height. That is, given h, to find a tree T' such that $height(T') \le h$ and

$$B(T') = \min\{B(T) \, : \, T \text{ is a BST satisfying } height(T) \le h\}.$$

Do the following.

(a) For $1 \le i \le j \le n$ and $0 \le k \le h$, set

$$e[i, j : k] \quad = \quad \begin{array}{l} \text{the minimum cost of any BST } T \\ \text{on } a_i, \ldots, a_j \text{ with } height(T) \le k \end{array} \quad . \qquad (2)$$

Give a Dynamic Programming recurrence equation for $e[i, j : k]$. Don't forget to provide the initial conditions.

(b) Justify (prove) the correctness of the recurrence relation from part (a).

You may use facts from the class lecture notes in your proof. If you do so, first explicitly provide the statement of the fact from the lecture notes. This statement should be in a paragraph separate from the rest of your proof and explicitly attribute the statement to the class notes.

(c) Give documented pseudocode for an $O(n^3 h)$ time algorithm to calculate $e[1, n : h]$.

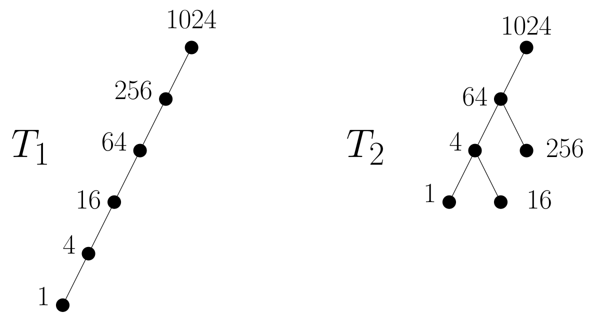
Note that $e[1, n : h]$ is the minimum cost of a BST on all the items with height $\le h$.

Your documentation should make clear what each section is doing.

(d) Explain why your algorithm runs in the required time.

Rules, Recommendations and Hints:

- As an example, consider the input $f(a_i) = 4^{i-1}$ for $i = 1, 2, 3, 4, 5, 6$, Tree T_1 is the min-cost tree BST for the given frequencies. It has height 5. T_2 is the min cost tree BST of height at most 3.



- Note that a binary tree with n total nodes must have height at least $\lfloor \log_2 n \rfloor$. For example, any tree with 5 nodes must have depth at least $\lfloor \log_2 5 \rfloor = 2$. So, the problem implicitly assumes that $h \geq \lfloor \log_2 n \rfloor$.
- If a tree does not exist, return ∞. For example, from the comment above, your table values should satisfy

$$e[i, j : k] = \infty$$

when $k < \lfloor \log_2(j - i + 1) \rfloor$.
As a special case, $e[i, j : 0] = \infty$ when $i < j$.
- Your pseudocode may use the convention that if x is any real number, $\infty + x = \infty$ and $\min(\infty, x) = x$.

P5: [35 pts] **Dynamic Programming**

The input to this problem is a directed "triangle graph" $G = (V, E)$ with h levels. It contains $n = h(h+1)/2$ nodes

$$\begin{aligned} V &= \{(i,j) \,:\, i \in [1,h],\ j \in [1,i]\} \\ E &= \bigcup_{\substack{(i,j)\in V \\ i<h}} \big\{ \big((i,j),(i+1,j)\big), \big((i,j),(i+1,j+1)\big) \big\} \end{aligned}$$

Intuitively, node (i,j) on level $i < h$ points to two nodes on level $i+1$; $(i+1,j)$ to its left (**L**) and $(i+1,j+1)$ to its right(**R**).

Each node in the tree has a given positive integral value, $A[i,j]$.

A *path* is a walk down the graph starting at $(1,1)$. It is described by a length $j < h$ string $w_1, w_2, \ldots, w_j \in \{L, R\}^j$. The path starts at the top node $(1,1)$ and, from level i it walks down to level $i+1$ either by going LEFT (if $w_i = L$) or RIGHT (if $w_i = R$).

- The *value* of a path is the sum of the values on the path.
- A *turn* on a path is a L followed by an R or a R followed by an L. For example $LLLRRRRLRR$ has three turns.
- A path is *balanced* if the number of even integers on the path is equal to the number of odd integers on the path.

Solve the following three problems using dynamic programming. The solutions for each of (a), (b) and (c) should start on a new page.

(a) Find the maximum value of a path in the graph.
Your algorithm should run in $O(n)$ time.

(b) Find the maximum value of a balanced path in the graph.
Your algorithm should run in $O(n^{3/2})$ time.

(c) Find the maximum value of a path containing *exactly* $\lfloor h/2 \rfloor$ turns in the graph. Your algorithm should run in $O(n^{3/2})$ time.

Each one of (a), (b), (c) should be split into the following five separate parts

(i) Give the recurrence upon which your DP algorithm is based.
Before stating the recurrence, define (using English and math symbols) the meaning of each table entry. Don't forget to completely specify the initial conditions of your recurrence relation.

(ii) Explain how, after filling in your table, you can use the information in the table to solve the stated problem.

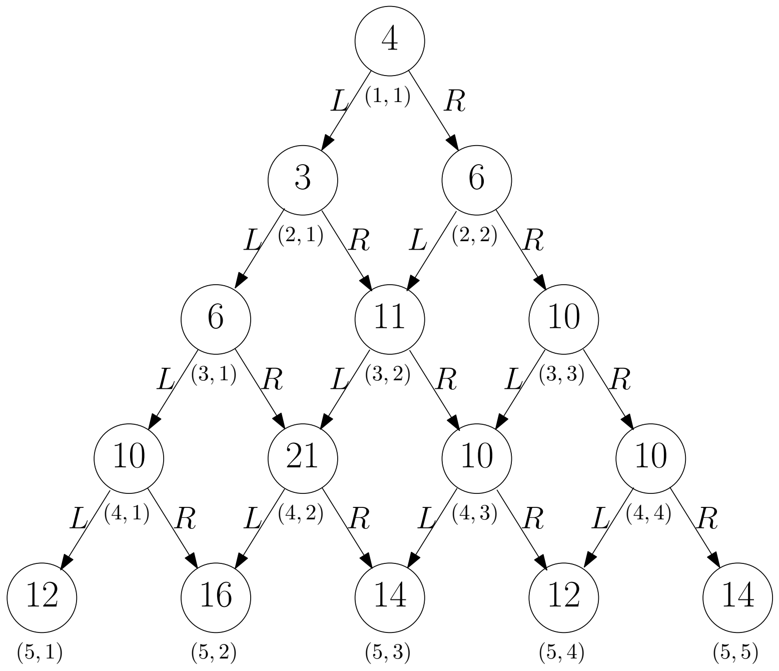
(iii) Justify (prove) the correctness of the recurrence relation from part (i).

(iv) Give documented psuedocode for your algorithm.

(v) Explain why your algorithm runs in the required time.

Rules, Recommendations and Hints:

- The diagram below illustrates the problem. The $A[\,]$ values are inside the nodes.



  (a) The max value path in the graph is $RLLL$, which has value $4 + 6 + 11 + 21 + 16 = 58$.

  (b) The max value balanced path in the graph is RLL, which has value $4 + 6 + 11 + 21 = 42$.

  (c) The max value path in the graph containing EXACTLY $\lfloor h/2 \rfloor = \lfloor 5/2 \rfloor = 2$ turns is $RLLR$ which has value $4+6+11+21+14 = 56$.

- Paths can have length at most $h - 1$. Balanced paths must have odd length.
- If a path does not exist, return $-\infty$. For example, if no balanced path exists in the graph, the answer to (b) would be $-\infty$. Your code may use the convention that if x is any integer, $-\infty + x = -\infty$ and $\max(-\infty, x) = x$.
- The main goal of this problem is figuring out the appropriate subproblems to use in the DP recurrences.

- For part (i) of each subproblem, you need to first define what the dynamic programming variables mean. That is, you need to provide a statement similar to Equations (2) and (3) in problems 4 and 6. You then give the recurrence.
- For (b) consider defining the "balance" of a path to be the number of even entries on the path minus the number of odd entires on the path. A "balanced" path would then be a path with balance equal to zero.

  Have your DP recurrences include a parameter denoting the balance of the paths.
- Hint. For (c) your recurrence should include a parameter counting the number of turns on a path. It might also need some other information.

P6: [35 pts] **Edit Distance**

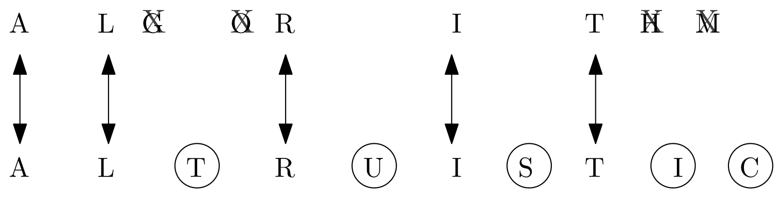
In this problem you must describe a dynamic programming algorithm for the *minimum edit distance* problem.

Background: The goal of the algorithm is to find a way to transform a "source" string $x[1 \dots m]$ into a new "target" string $y[1 \dots n]$ using any sequence of operations, *operating on the source string from left to right.* The *copy* operation copies the first remaining character in the source string to the target string, and deletes it from the source string. The *insert* operation adds one character to the end of the current target string. The *delete* operation deletes the first remaining character from the source string.

For example, one way to transform the source string `algorithm` to the target string `altruistic` is to use the following sequence of operations.

Operation	Target string	Source string
copy `a`	`a`	`lgorithm`
copy `l`	`al`	`gorithm`
delete `g`	`al`	`orithm`
insert `t`	`alt`	`orithm`
delete `o`	`alt`	`rithm`
copy `r`	`altr`	`ithm`
insert `u`	`altru`	`ithm`
copy `i`	`altrui`	`thm`
insert `s`	`altruis`	`thm`
copy `t`	`altruist`	`hm`
delete `h`	`altruist`	`m`
insert `i`	`altruisti`	`m`
delete `m`	`altruisti`	
insert `c`	`altruistic`	

The operations can be visualized as follows, with the bi-directional arrows signifying a copy, the circles an insert and the crosses a delete.



Each of the operations has an associated cost, **C** for copy, **I** for insert, and **D** for delete. The cost of a given sequence of transformation operations is the sum of the costs of the individual operations in the sequence. For the example above, the cost of converting `algorithm` to `altruistic` using the given set of operations is $5\mathbf{C} + 5\mathbf{I} + 4\mathbf{D}$. If $\mathbf{C} = 1$, $\mathbf{I} = 2$ and $\mathbf{D} = 3$ the cost of the edit would be $5 + 10 + 12 = 27$.

Given two sequences $x[1 \ldots m]$ and $y[1 \ldots n]$ and a given set of operation costs $\mathbf{C}$, $\mathbf{I}$, and $\mathbf{D}$, the *minimum edit distance* from x to y is the cost of the *least expensive transformation sequence* that converts x to y.

(a) Set $X[i] = x[1..i]$ and $Y[j] = y[1..j]$. Define the cost matrix as

$$D[i,j] = \text{minimum edit distance to transform } X[i] \text{ to } Y[j] \qquad (3)$$

Give a Dynamic Programming recurrence equation for $D[i,j]$. Don't forget to provide the initial conditions.

(b) Justify (prove) the correctness of the recurrence relation from part (a).

(c) Give documented pseudocode for an $O(mn)$ time algorithm to calculate $D[m,n]$. Your documentation should make clear what each section is doing.

(d) Explain why your algorithm runs in the required time.

Rules, Recommendations and Hints:

- Note that if $x[i] = y[j]$ then, physically, a *copy* of $x[i]$ to $y[j]$ could be replaced by *delete* $x[i]$ followed by *insert* $y[j]$. To avoid dealing with this situation you should assume that $\mathbf{C} < \mathbf{D} + \mathbf{I}$.
- The algorithm is very related to the algorithm for the longest common subsequence problem taught in class.
- Consider the last operation in the transformation from $X[i] = x[1 \ldots i]$ to $Y[j] = y[1 \ldots ..j]$. How does the cost of the entire transformation depend on the cost of the last operation?
- Comment: The minimum edit distance between two genes is one measure used in bioinformatics to determine how "close" two genes are to each other.

P7: [25 pts] **Greedy Algorithms**
Give an $O(n \log n)$ algorithm for assigning location of n files on a magnetic tapes that minimizes the expected time necessary to read a file. As input you are given the size of the files and the probability that each file will be accessed.

Background: Files used to be stored on magnetic tapes rather than disks. Reading a file from tape is not like reading a file from a random access disk. On a tape, the files are laid out sequentially, one after another. To access a file on a tape we first must scan all the other files preceding the file, which takes a significant amount of time. The ordering of the files on the tape therefore defines how long it takes to read any particular file.

More explicitly, an ordering of the files will just be the ordering of how to store the files on the tape. For example, the ordering $(3, 1, 2, 4)$ means that file 1 will be placed as the 2^{nd} file, file 2 will be placed as the 3^{rd} file, file 3 will be placed as the 1^{st} file and file 4 will be placed as the 4^{th} file.

Suppose $L[i]$ is the size of file i. The time T_i it takes to read file i is the sum of the sizes of the files located before file i plus the size of file i. In the example above, if $L = [2, 6, 4, 8]$, then

$$\begin{aligned}
T_3 &= L[3] = 4, \\
T_1 &= L[3] + L[1] = 4 + 2 = 6 \\
T_2 &= L[3] + L[1] + L[2] = 4 + 2 + 6 = 12 \\
T_4 &= L[3] + L[1] + L[2] + L[4] = 4 + 2 + 6 + 8 = 20
\end{aligned}$$

Suppose that you know that file i will be accessed with probability p_i and the ordering in which the files are on the tape. The expected (average) time to read a file will be

$$\sum_i p_i T_i.$$

To continue the example above, if the probabilities were $p = [\frac{1}{6}, \frac{1}{12}, \frac{1}{4}, \frac{1}{2}]$ then the expected read time would be

$$\frac{1}{6} \cdot 6 + \frac{1}{12} \cdot 12 + \frac{1}{4} \cdot 4 + \frac{1}{2} \cdot 20 = 13.$$

On the other hand, if we had the same probabilities but the files were in the order $(1, 3, 4, 2)$ then the reading times would be

$$\begin{aligned}
T_1' &= L[1] = 2 \\
T_3' &= L[1] + L[3] = 2 + 4 = 6 \\
T_4' &= L[1] + L[3] + L[4] = 2 + 4 + 8 = 14 \\
T_2' &= L[1] + L[3] + L[4] + L[2] = 2 + 4 + 8 + 6 = 20
\end{aligned}$$

and the expected read time would be

$$\frac{1}{6} \cdot 2 + \frac{1}{12} \cdot 20 + \frac{1}{4} \cdot 6 + \frac{1}{2} \cdot 14 = 10.5,$$

which is less than the time for the previous ordering.

The problem:

Input is an array of $L[1..n]$ of the n files sizes and a list $p_1, p_2, \ldots, p_n$ of the probability of accessing each file.

The problem is to design an $O(n \log n)$ algorithm that outputs an optimal ordering of the files on the tape, i.e., an ordering that minimizes the expected time to read a file.

To do this answer the following questions:

(a) First assume that all the files are equally likely, i.e., for all files $p_i = \frac{1}{n}$. Prove that in this case, inserting the files in order of increasing size gives an optimal solution.
For example, if $p_i = 0.2$ for all i and $L[1..5] = [2, 5, 7, 1, 4]$ they should be inserted in the increasing size order $1, 2, 4, 5, 7$.

Note that this property would lead to an $O(n \log n)$ algorithm for solving the problem in the equal probability case.

(b) Show that in the general case, sorting the files by size and inserting them on the tape in order of increasing size does not have to give an optimal solution.

More specifically, describe an input of size 3 for which ordering by increasing size of the file does not give the correct solution. To do this, explicitly calculate the solution cost when the files are sorted by increasing size and then show a cheaper solution.

(c) Describe a rule that does solve the problem.

That is, find a different rule for ordering the items in the general case such that if you sort the items in that order you WILL get an optimal solution (this rule should work for all inputs).

(d) Provide a formal proof that the rule that you gave in part (c) outputs an optimal ordering.

(This must be formal. Make every assumption used explicit, and justify every step in your argument. We will deduct points for ambiguity.)

(e) Explain how this gives you an $O(n \log n)$ greedy algorithm for solving the problem.

(To get full credits, your ordering must lead to an $O(n \log n)$ solution.)

P8: [20 pts] **Greedy Agorithms**

Consider a long river, along which n houses are located. You can think of this river as an x-axis; the house locations are given by their coordinates on this axis in a sorted order. The inputs are $x_1 < x_2 < \cdots < x_n$.

Your company wants to place cell phone base stations along the river, so that every house is within 10 kilometers of one of the base stations.

(a) Give an $O(n)$-time algorithm greedy algorithm that minimizes the number of base stations used.

Provide both documented pseudocode and an explanation in words as to what it does.

(b) Provide a formal proof of your algorithm's correctness, i.e., that it outputs a minimal size solution.

(This must be formal. Make every assumption used explicit and justify every step in your argument. We will deduct points for ambiguity.)

Rules, Recommendations and Hints:

- Hint. See Tutorial GY1 for a related problem.
- Your solution for this problem should be a greedy algorithm. That is, it should create a solution set of S of base stations by scanning through the input data, one house at a time. Every time it looks at a new house, it should decide whether or not to add a new base station to S (and if so, where). Once a base station is added to S, that base station is never removed.
- The algorithm should be simple. The real work is in proving correctness in (b).
- Note that the solution to problem 9, restricted to $t = 1$, $c_1 = 1$ and $k_1 = 10$, actually also solves this problem in $O(n)$ time. But, that solution will not be a greedy solution and can not be used here.

P9: [25 pts] **Dynamic Programming**

Now consider a modification of the previous problem. You still have a long river on which house locations are given by their coordinates on this axis in a sorted order. The inputs are still $x_1 < x_2 < \cdots < x_n$.

Your company still wants to place cell phone base stations along the river to cover every house. The difference is that you now have t different types of base stations. A base station of type i costs c_i dollars to install and can cover all houses within k_i kilometers of its location. A base station of type i located at y is denoted by pair (y, i).

A house at location x is *covered* by base station (y, i), if $x \in [y - k_i, y + k_i]$.

A set

$$S = \{(y_1, i_1), (y_2, i_2), \ldots, (y_s, i_s)\}$$

is a *cover* of the input, if it covers all of the houses, i.e.,

$$\{x_1, \ldots, x_n\} \subset \bigcup_{j=1}^{s} \left[y_j - k_{i_j},\, y_j + k_{i_j}\right].$$

The *cost* of a set of base stations is $\mathit{cost}(S) = \sum_{j=1}^{s} c_{i_j}$.

Design an $O(tn)$ dynamic programming algorithm to find the cheapest (i.e., least expensive) cost of a set of base stations that covers all of the houses.

(a) Give the recurrence upon which your DP algorithm is based. Before stating the recurrence, define (using English and math symbols) the meaning of each entry. Don't forget to completely specify the initial conditions of your recurrence relation.

(b) Explain how, after filling in your table, you can use the information in your table to solve the stated problem.

(c) Prove the correctness of the recurrence relation from part (a).
(This must be formal. Make every assumption used explicit, and justify every step in your argument. We will deduct points for ambiguity.)

(d) Give documented psuedocode for your algorithm.

(e) Explain why your algorithm runs in the required time.

Rules, Recommendations and Hints:

- You may assume that $c_1 < c_2 < \cdots < c_t$ and $k_1 < k_2 < \cdots < k_t$.
- Your algorithm only needs to return the COST of the cheapest placement. You do not have to output where the base stations are placed.
- For part (a) you need to first define what the dynamic programming variables mean. That is, you need to provide a statement similar to Equations (2) and (3) in problems 4 and 6. You then give the recurrence.
- Hint. Consider the location of the "last" base station in an optimal solution.
- Hint. Reading tutorial DP2 might give you some ideas.
- Your algorithm may use $O(tn)$ space.
- Full marks require $O(tn)$ time algorithms. Algorithms that use $O(tn \log n)$ time will have points deducted. Algorithms that use more than $O(tn \log n)$ will be considered wrong.