CMPT125, Spring 2022

Homework Assignment 5
Due date: April 15, 2022

You need to add your solution to **assignment5.zip**.
Submit the file **assignment5.zip** to CourSys.

For this assignment you will create a project in C++ that implements a solver
for the Traveling Salesperson Problem (TSP).
- You will need to design your own classes.
- You will need to decide how to partition the classes into files.
- You will need to decide what goes into .hpp and what goes into .cpp.

You will also need to write your own main() in a separate file.
The main() will test your TSP solver on examples that you will create yourself.
See the provide

You may use an   ourse

The file Assignment5_supportFiles.zip c   p and .hpp files as
well as Makefile. You may modify any of

*** Make sure that Makefile creates the executable called **tspsolver**.***

Submit all your files in **assignment5.zip** to **Coursys**.

Good luck!

The Traveling Salesperson Problem

The input to the problem is a collection of *n* points in the plane. The points have int values.
The goal of the traveling salesperson problem is to find the shortest path that visits every point exactly once and returns to the starting point. That is, we are looking for a cycle in the graph that visits each vertex exactly once, such that the total length is as small as possible.

For example, suppose the input is the 4 points
A = (2,2), B = (2,6), C = (5,6), D = (5,9)
Then we may consider each of the following cycles through the 4 points.
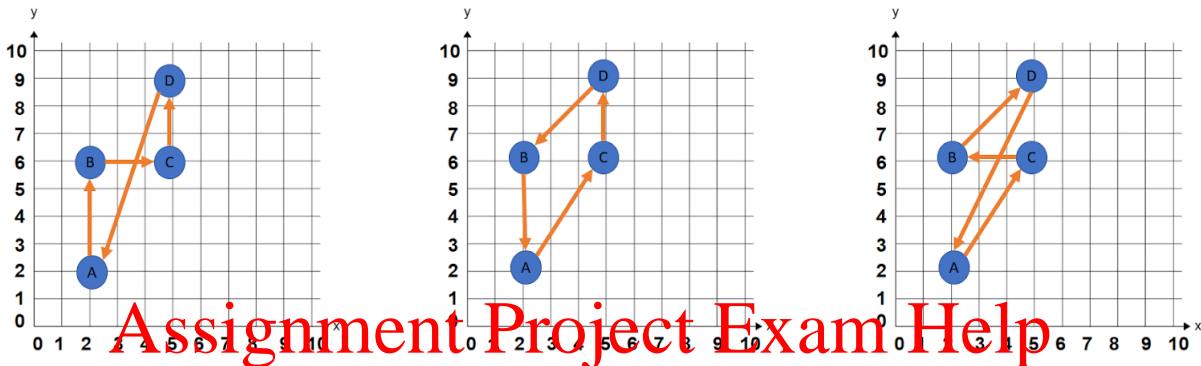
- The length o

  dist(A,B)+dis

- The length of the cycle in the middle ACDB (

  dist(A,C)+dist(C,D)+dist(D,B)+dist(B,A) = $\sqrt{3}$ ~~edu~~ $+4 \approx 16.243$

- The length of the cycle on the right ACBD (and back to A) is

  dist(A,C)+dist(C,B)+dist(B,D)+dist(D,A) = $\sqrt{3^2 + 4^2} + 3 + \sqrt{3^2 + 3^2} + \sqrt{3^2 + 7^2} \approx 19.858$

That is, among these three cycles, the better one is ACDB, whose length is 16.243...
In fact, this is the optimal solution for this input.

You will need to design your own classes in C++ to implement a solver for this problem.

Requirements:

The requirements for this project are the following [The exact implementation details are described below].

Storing the points:
You will need to write a class that stores a collection of points. You may use any data structure you want to do it (array, linked list or vector in C++)

Print the list of points:
Your solution should have a method that prints the list of all points. The format is up to you. For example, you can use the format
A = (2,2)
B = (2,6)
C = (5,6)
D = (5,9)

Drawing the points:
Your solution should have a method that draws the points on the screen.
No need to use anything the board suffices.
You may assume *for* d 20 and names of points
are single letters. For
- - - - - - - - - - -
- - - - - - - - D - -
- - - - - - - - - - -
- - - - - - - - - - -
- - B - - - - C - -
- - - - - - - - - - -
- - - - - - - - - - -
- - - - - - - - - - -
- - A - - - - - - -
- - - - - - - - - - -
- - - - - - - - - - -

TSPSolver:
This is the main part of this project. You will need to implement a heuristic algorithm that finds a solution to the TSP problem. For the algorithm see the part **TSP solver** below.

main():
You will need to write the main() function that will run several tests to check your solution.
Write a test for every part of your code. Write as many tests as you think are necessary.
At the very least, your tests should (1) provide inputs to the problem (2) print the list using the printList method (3) draw the points (4) run the solver and output the obtained solution: you need to print to screen both the order of the vertices in the cycle and the total length of the cycle.

TSP solver

The [Traveling Salesperson Problem](#) is [NP-complete](#). Without saying exactly what this means, this implies that we don't know an efficient algorithm that finds an optimal solution for every input.

Instead of trying to find an optimal solution, your TSPsolver will need to implement the following heuristic algorithm, which we will call "***smallest increase heuristic***".

Smallest increase heuristic:

- The input is a list L[0...n-1] of n≥3 points.
- The algorithm starts with the cycle consisting of only the points L[0], L[1], L[2].
- In the first iteration the algorithm takes the point L[3], and adds it to the current cycle in the location that minimizes the increase in the length.
- In the second iteration the algorithm takes the point L[4], and adds it to the current cycle in the location that minimizes the increase in the length.
- And so on. When adding the point L[i], we add it to the cycle between C[j] and C[j+1] so that `dist(C[j],L[i]) + dist(L[i],C[j+1]) - dist(C[j+1],C[j])` is minimized.

Assignment Project Exam Help

** This heuristic is not guaranteed to find an optimal solution. Instead, it chooses the position for each point in the cy

https://eduassistpro.github.io/

Example:

Consider the example above with 4 points, and sup   ,D,A].

Add WeChat edu_assist_pro

1. We start with the cycle consisting of the poin
2. In the first iteration, we choose where to add

   ➢ If we add A between C and B, then the increase will be

   dist(C,A) +dist(A,B) - dist(B,C) = $\sqrt{3^2 + 4^2} + 4 - 3 = 6$

   ➢ If we add A between B and D, then the increase will be

   dist(B,A) +dist(A,D) - dist(D,B) = $4 + \sqrt{3^2 + 7^2} - \sqrt{3^2 + 3^2} = 7.3731...$

   ➢ If we add A between D and C, then the increase will be

   dist(D,A) +dist(A,C) - dist(C,D) = $\sqrt{3^2 + 7^2} + \sqrt{3^2 + 4^2} - 3 = 9.6157...$

3. Therefore, it is best to add A between C and B.
4. The resulting cycle will be CABD (and back to C)

Implementing classes:

There are no specific requirements about which classes you need to implement. However, your solution must meet the requirements described on page 3.

Here are some suggestions you may use. They are also provided in Assignment5_supportFiles.zip

These are examples only. You may choose a completely different implementation if you want.

```cpp
// the class represents a point in the grid
class Point {
private:
    string name;
    int x;
    int y;
public:
    // computes the distance between this and the other point
    float getDistance(const Point &other)
    // also contains getters and setters, see .hpp for details
}

// the class stores an ordered list of points
// used to store the input to the problem
// may also be used to store a partial solution to the TSP problem
class ListOfPoints {
    // some data structure to store the points
public:
    // adds a newPt after a point with the given name/given index in the list
    void addAfter(Point& newPt, string name);
    void addAfter(Point& newPt, int index); // you decide which one you prefer
    // adds a new point to the end of the list
    void addToBack(Point& newPt);
    // gets a copy of the point from the list at index i
    Point& get
    // gets th
    int getSiz
    // prints
    void print
    // draws the points
    void draw() const;
}

// stores a cycle that traverses all points in some order
class TSPCycle : public ListOfPoints {
public:
    float getLength() const // returns the total length of the cycle
}

// implementation of the TSP solver
class TSPSolver {
private:
    // may hold the following members:
    ListOfPoints m_list;
    TSPCycle m_solution; // you may remove/modify these in any way you want
public:
    // an example of a constructor - gets list of points
    TSPSolver(ListOfPoints &list);
    // solves the problem, and stores the solution
    void solve();
    // returns the solution obtained by the solve() method.
    TSPCycle& getSolution();
}
```

*Other heuristics [NOT FOR SUBMISSION]***:**
You may try implementing several other heuristics if you want.
For example, consider the following two greedy ideas:

● *Nearest last point heuristic:* The algorithm starts with a path consisting of one point. In each iteration the algorithm builds a path P by adding to it one point at a time. Specifically, the algorithm finds a point that has not been added to P yet, which is the closest to the last point in P (breaking ties arbitrarily). This point is added to P as the last point. In the end the last point is connected to the first point to close the cycle.

● *Nearest existing point heuristic:* The algorithm starts with the path consisting of L[0]. In each iteration take the next point L[i], and add it to the current path after the point to which it is closest, breaking ties arbitrarily. When all points have been added, close the cycle by connecting the last point in the path to the first point.

● *Extended smallest increase heuristic:* Same as what you need to do for the assignment, but in each iteration the algorithm chooses *which point is best to choose among the remaining points*, and where to add it in the cycle.