Project 4

EECS 281

Agenda

- Graphs and Minimum Spanning Trees
  - Prim’s Algorithm
  - Kruskal’s Algorithm
- The Travelling Salesperson problem
  - Optimal solution algorithm
  - Fast but not optimal algorithm
- Project 4 FAQ

Order of Solution

- Do them in the order given:
  - MST
  - FASTTSP
  - OPTTSP
- Why? OPTTSP can use the first two
  - FASTTSP: best so far
  - MST: used for lower bound

Visualizing Results

- Use the visualization tool
- Only available on Autograder 2
  - AG1 runs the SQL server
  - We didn’t want to add more for it to do
- https://g281-2.eecs.umich.edu/p4viz/

Graphs

- A set of objects where some/all of them are connected by links
  - Use cases?

Graphs

- Different types of graphs
  - Directed/Undirected
  - Weighted/Unweighted
  - Multigraph
  - ..

Graphs

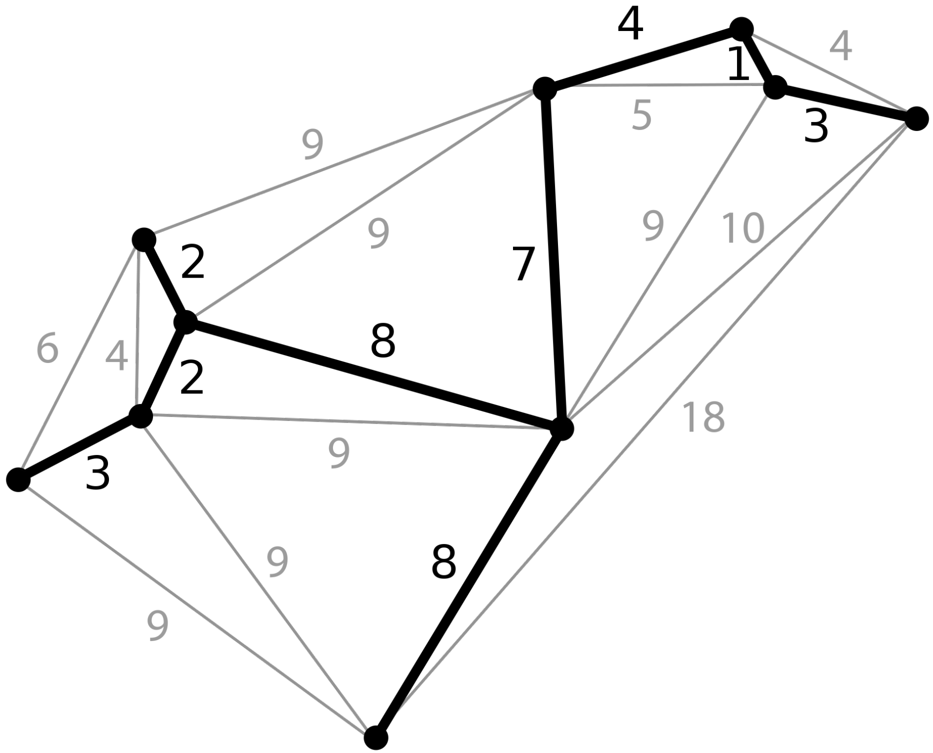
- Different types of graphs
  - Directed/Undirected
  - Weighted/Unweighted
  - Multigraph
  - ..

- Know these terms for the exam!

Minimum Spanning Tree

- <u>Problem</u>: Given a graph of cities, devise a minimum cost method (in terms of length of path constructed) of connecting them all together.

Minimum Spanning Tree



Minimum Spanning Tree

- This is not NP-hard.
  - It is much easier to solve this.

- For more see EECS 376

Minimum Spanning Tree

- Given a MST of a graph G and a point A not in the graph. Construct an MST with the graph formed by joining every vertex in G with A.

Minimum Spanning Tree

- Given a MST of a graph G and a point A not in the graph. Construct an MST with the graph formed by joining every vertex in G with A.

- Modify this algorithm to produce an MST of a whole graph.

Minimum Spanning Tree

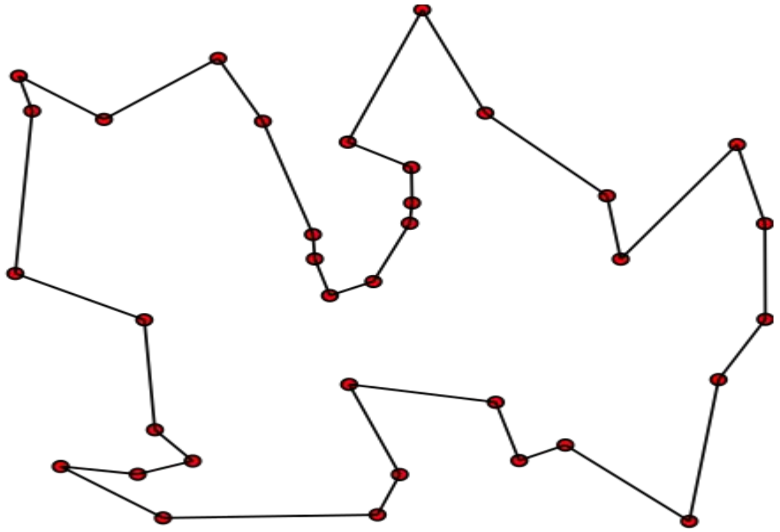
- Prim's Algorithm
  a. Mark all nodes unvisited, distance ∞, no previous
  b. Pick a starting point; change its distance to 0
  c. While there are unvisited nodes:
     - Connect one of the visited nodes to an unvisited node with the shortest distance possible.
     - Mark the new node visited
     - Update distance of any node adjacent to that node

Travelling Salesperson

- Problem: Given a graph, find the shortest path to visit all nodes in the graph and come back to the starting position

Travelling Salesperson

- What is the starting point? Does it matter?



Travelling Salesperson

- Problem: Given a graph, find the shortest path to visit all nodes in the graph and come back to the starting position
- This is an NP-hard problem
  - NP-hard problems can be even more difficult than NP-complete problems! (see EECS 376)

Travelling Salesperson

- Problem: Given a graph, find the shortest path to visit all nodes in the graph and come back to the starting position

- If the graph is unweighted and complete then how can we solve this problem?

Travelling Salesperson

- Problem: Given a graph, find the shortest path to visit all nodes in the graph and come back to the starting position

- Now consider a weighted directed graph. How can we solve this problem?
    - One possible solution: Consider all possible routes! or in other words, Brute force!

Brute force

- Guess the password: A user on Facebook can have a 4 letter password comprised of ASCII characters. Guess his password. You have unlimited attempts.

Brute force

- Guess the password: A user on Facebook can have a 4 letter password comprised of ASCII characters. Guess his password. You have unlimited attempts.

- Guess all possible permutations!
    - How many permutations will you consider?

Brute force

- Inference: Brute force algorithms are usually the worst possible solution to a problem.

Brute force

- Inference: Brute force algorithms are usually the worst possible solution to a problem.
  - But we will optimize

Brute force

- Guess the password: A user on Facebook can have a 4 letter password comprised of ASCII characters. Guess his password. You have unlimited attempts.
- You deduce somehow that the third letter can only be an ‘a’ or a ‘c’.

Brute force

- Guess the password: A user on Facebook can have a 4 letter password comprised of ASCII characters. Guess his password. You have unlimited attempts.
- You deduce somehow that the third letter can only be an ‘a’ or a ‘c’.
  - How much better is this than the previous solution?

Brute force

- Guess the password: A user on Facebook can have a 4 letter password comprised of ASCII characters. Guess his password. You have unlimited attempts.
- You deduce somehow that the third letter can only be an ‘a’.
    - Now how many cases would you consider?

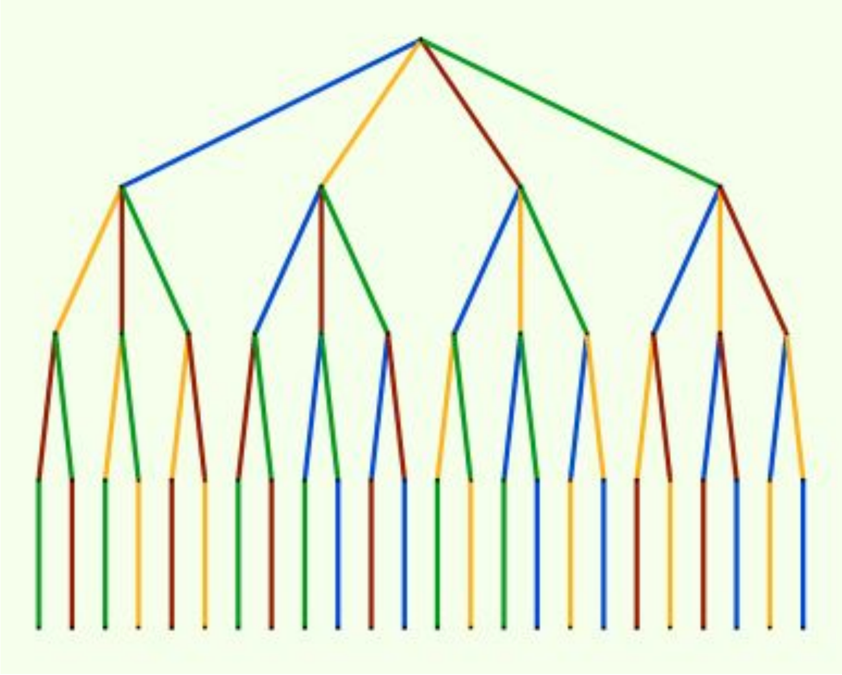
Brute force

- This is the essence of the branch and bound optimization. You think smartly and eliminate multiple possibilities to get better runtime.

Travelling Salesperson

- How to generate all possible routes from point A to point B in a graph?
  - Randomly connect edges?

Travelling Salesperson



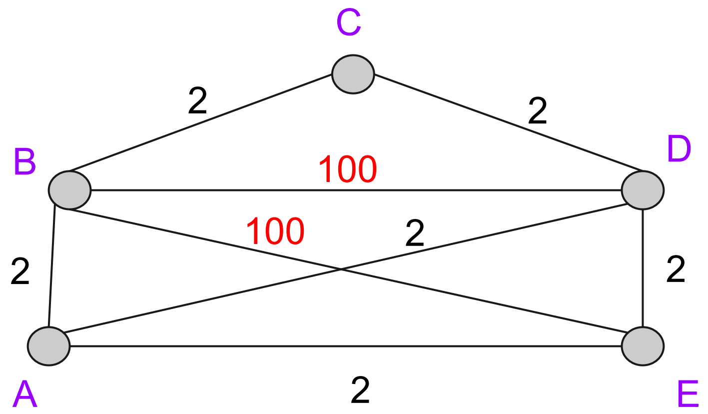
Travelling Salesperson

- How can we eliminate some unnecessary permutations while brute forcing the TSP problem?

Travelling Salesperson

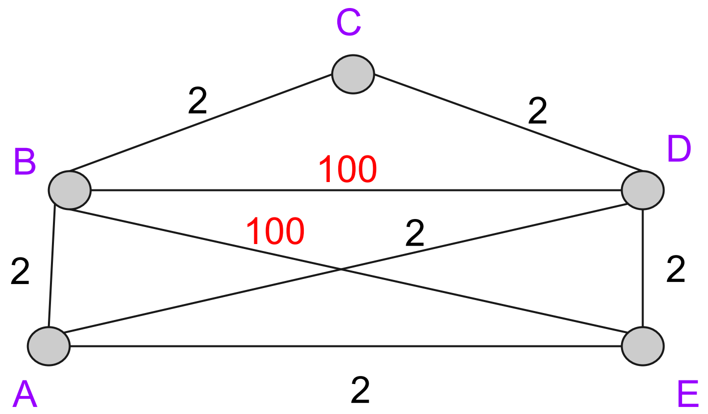
- How can we eliminate some unnecessary permutations while brute forcing the TSP problem?
  - Keep track of previous best. If while generating permutations you exceed previous best. Discard current solution and move on to the next.

Travelling Salesperson



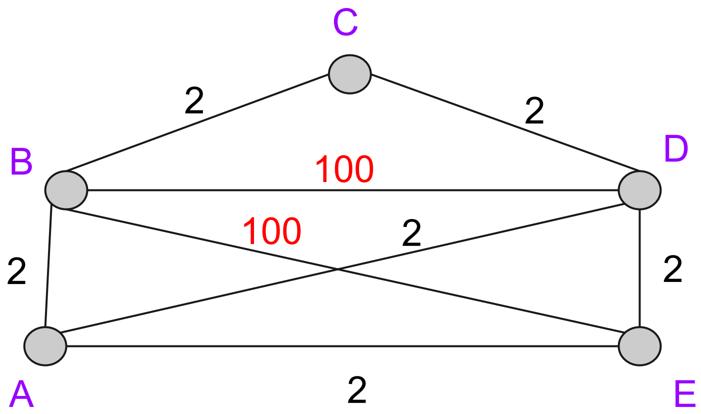
- What is the optimal path here?

Travelling Salesperson



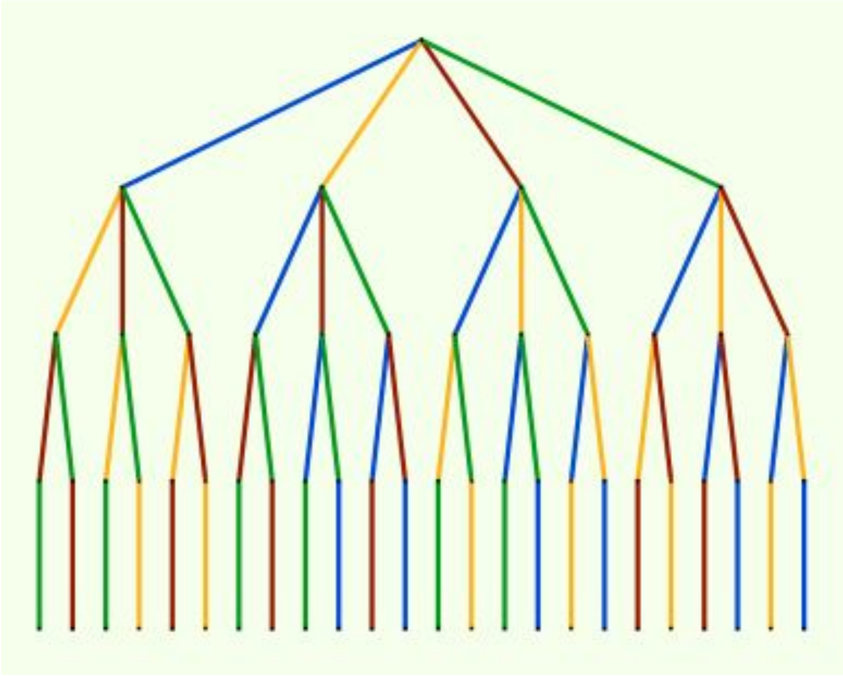
- What is the optimal path here?
  - Around the edges.

Travelling Salesperson



- Eliminate
  - A->B->E…..
  - A->B->D…..

Travelling Salesperson



Travelling Salesperson

```
template <typename T>
void genPerms(vector<T> &path, size_t permLength) {
  if (path.size() == permLength) {
    // Do something with the path
    return;
  } // if
  if (!promising(path,  permLength)) // Add custom logic in promising()
      return;
  for (size_t i = permLength; i < path.size(); ++i) {
      swap(path[permLength], path[i]);
      genPerms(path, permLength + 1);
      swap(path[permLength], path[i]);
  } // for
} // genPerms()
```

OPTTSP ~~💘~~ MST

- Can we somehow eliminate a branch of the tree that starts out poorly, and will thus never lead to a solution that's better than our best so far?
    - Estimate cost of the remaining k nodes
    - Estimate must be faster than O(k!)
    - Big hint for p4

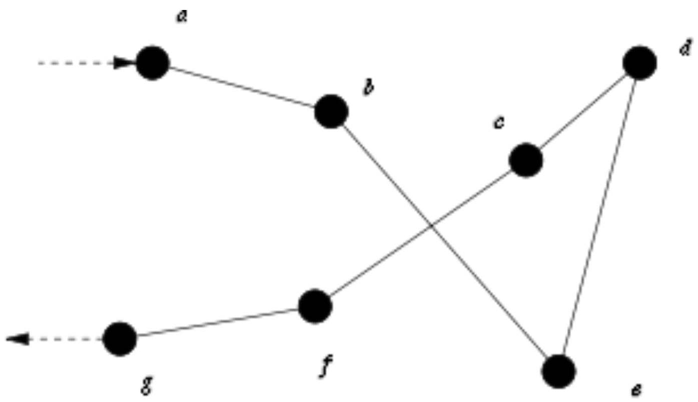
Fast-TSP

- It is inefficient even for a supercomputer to solve the TSP problem, so most people estimate a solution

Fast-TSP

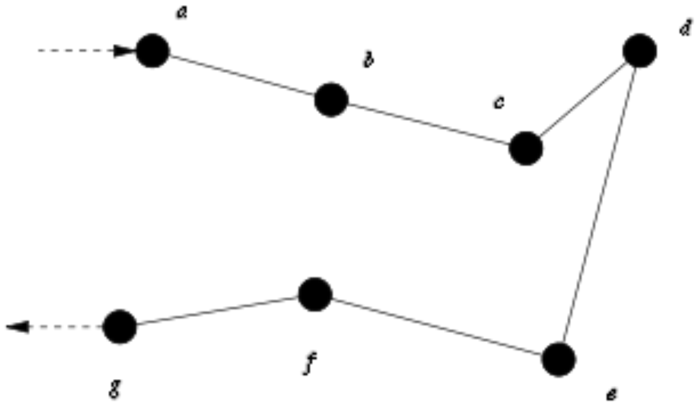
- It is inefficient even for a supercomputer to solve the TSP problem, so most people estimate a solution
    - Solve in a greedy manner, i.e. add the closest point to the current point you’re on and repeat
    - This is not the only, or even the best way, but it works fairly well

Fast-TSP



- Does this look like an efficient tour for our salesperson?

Fast-TSP

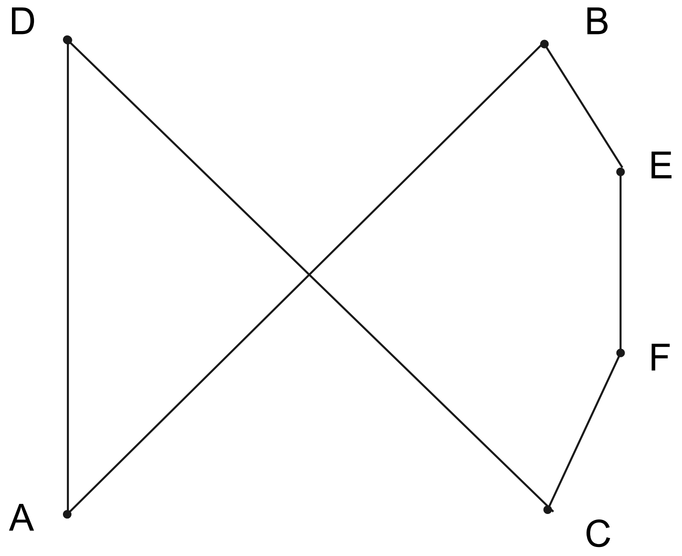


- This looks better

Improving Heuristics

- Suppose you come up with a heuristic for the FASTTSP, and your solution path is too long to get full credit, two options:
  - Change the heuristic
  - Add 2-Opt
- Be willing to try out other heuristics!
  - Greedy + 2-Opt will NOT earn you all the points for FASTTSP, but it will earn most of them

Suppose Starting Path...

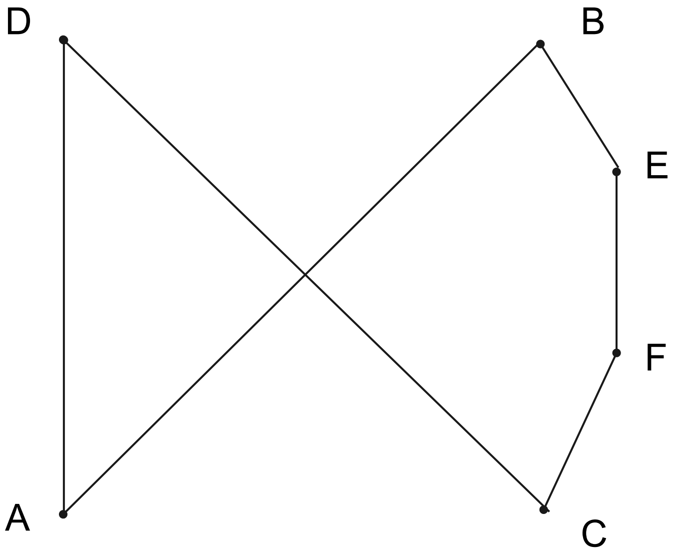


Current path:

A - B - E - F - C - D (- A)

The (- A) means that a full cycle would include A, but we could just keep track of A - B - E - F - C - D

2-Opt Time



Suppose we're considering optimizing A-B and C-D

A-B length = 1.4; C-D = 1.4

Total = 2.8

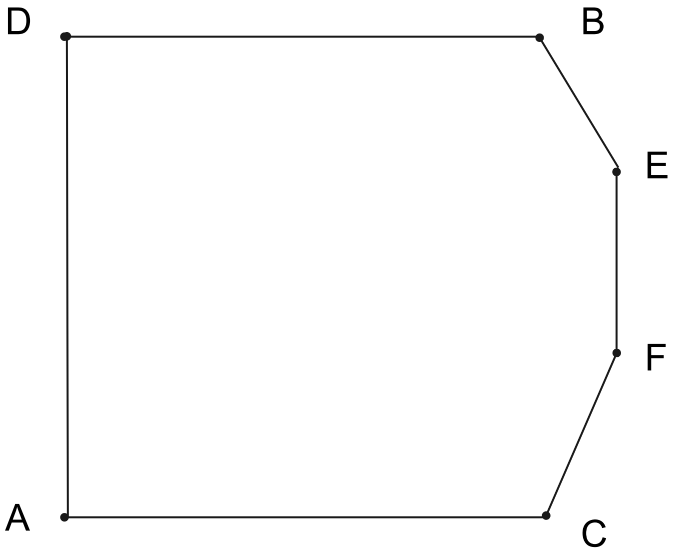
Replace with:

A-C = 1; B-D = 1

Total = 2

Good savings = swap

2-Opt Time



Revised path:

A - C - F - E - B - D (- A)

Path Changes

- Notice that the path has changed from:

A - B - E - F - C - D (- A)

- To:

A - C - F - E - B - D (- A)

- The entire middle has reversed order!

B - E - F - C Has become C - F - E - B

Run Through All Possible

- Always check adjacent pairs, compared to all other adjacent pairs
- As soon as you see an improvement, make it
- Pick up where you left off (think in terms of indices into the path)
- $O(V^2)$

Project 4 Tips

- Given vertices as ordered pairs in the x-y plane, how will you find out which line segment is smaller?
- For example:
  - v1: {3, 3} v2: {6, 10} v3: {8, 8}
- Which is shorter, v1 to v2, or v1 to v3?
- How do you KNOW, without a calculator?

Project 4 Tips

- The idea from the previous slide works when comparing one line segment to another, NOT when summing up a set of line segments!
- When computing Euclidean distance don't use `pow()`; multiply or use `sqrt()` as appropriate

Problem Size / Distance Matrix

- In the MST and FASTTSP portions, the graph might have tens of thousands of vertices
  - Is there enough memory available to store a distance matrix?
  - Consider 50,000 vertices, 8 bytes per double
- In OPTTSP, problem size limited to < 40 nodes
  - Room for distance matrix, and faster if one exists

Functors!

- Each part can use a different functor for calculating distance between two points
- In MST, what is distance between a “normal” cage and one fully in the wild animal area?
- In OPTTSP, there are so few nodes that you can pre-compute all possible distances
  - Functor can store the distance matrix as member

Project 4 Tips

- You will be given graphs in P4 to execute algorithms on. How would you store them in memory?

Project 4 Tips

- Why are we suggesting Prim's algorithm over Kruskal's?
- Is our graph dense in the MST part?