Presentation Link:

https://youtu.be/0SUq0SPmJUM

Github Link:

https://github.com/zhouliupku/cs5800_final_project

Algorithms to Solve Real Travel Planning Problems

Zhou Liu, Jifan Xie, Jing Ming

CS5800 final Project

Outline

- **Question to answer**: Travel planning under different situations
- **Problem 1**: Path Existence Problem
- **Problem 2**: Single Source Shortest Path Problem
- **Problem 3**: All Pairs Shortest Problem
- **Problem 4**: Travelling Salesman Problem
    - Dynamic Programming Solution
    - Greedy Solution
    - Approximation Solution

Problem 1: Path Existence Problem

Situation: Decide whether there exist a path between 2 spots

Model: G = (V, E)

- edge representation? -> travelling time / distance / money / gasoline / electricity
  - directed or undirected? undirected
  - positive weighted or negative weighted?

Solution to Problem 1

Algorithm: BFS or DFS

Implementation:

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1                 // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]           // explore edge (u, v)
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK                 // blacken u; it is finished
time = time + 1
u.f = time
```

Time complexity: O(V+E)

Problem 2: Single Source Shortest Path Problem

Situation: Plan a trip around home with the minimum cost

Model: G = (V, E)

- edge representation? -> travelling time / distance / money / gasoline / electricity
  - directed or undirected?
  - positive weighted or negative weighted?

Solution to Problem 2

	Directed	Weighted	Time Complexity
Dijkstra's algorithm	**Directed and Undirected**	non-negative	Array: $O(V^2)$ Binary Heap: $O(E \log V + V \log V)$ Fibonacci Heap: $\mathbf{O(E + V \log V)}$
Bellman-Ford algorithm	Directed or Undirected with non-negative weights	**allow negative weights but not allow negative cycles**	$O(V * E)$

Problem 3: All Pairs Shortest Path Problem

Situation: Calculate shortest path between each pairs of cities for travelling convience

Model: G = (V, E)

- edge representation? -> travelling time / distance / cost
- directed or undirected? -> directed
- positive weighted or negative weighted? -> positive weighted

Solution to Problem 3

Algorithm: Floyd Marshall Algorithm

Implementation: Dynamic Programming

For each pair, try whether it's distance can be reduced by cities1, cites2, „„ citiesN.

Time complexity: $O(N^3)$

```
for k in range(n):
    for i in range(n):
        for j in range(n):
            if dp[i][j] > dp[i][k] + dp[k][j]:
                dp[i][j] = dp[i][k] + dp[k][j]
```

Problem 4: Travelling Salesman Problem

Situation: planning a tour that visits all spots exactly once except the start vertex.

TSP is an NP-complete problem.

Precise Solution

1. **Brute Force**

   Time complexity: O(N!) time

2. **Dynamic Programming**

   Time complexity: $O(N^2 * 2^N)$ time

   Basic intuition: If we found the shortest path from 0, 1, 2, 3… to N, and back to 0, the path from 1, 2, …, to N and back to 0 is also a shortest path. -> Subproblem

Dynamic Programming Solution to TSP

1. **DP matrix**

   dp[i][V]:

   i: a vertex in the graph

   V: a set of vertices

   The shortes path value that:

   Start from i

   Visiting all vertices in V for one and only one time

   Back to vertex 0

Dynamic Programming Solution to TSP

2. DP state transition equation

Let C_{ij} represents the distance from vertex i to vertex j.

If V $= \emptyset$ and i $\neq$ s, dp[i][V] $= C_{is}$

If V $\neq \emptyset$ and k $\in$ V, dp[i][V] $=$ min(C_{ik} + dp[k][V - $\{k\}$])

Dynamic Programming Solution to TSP

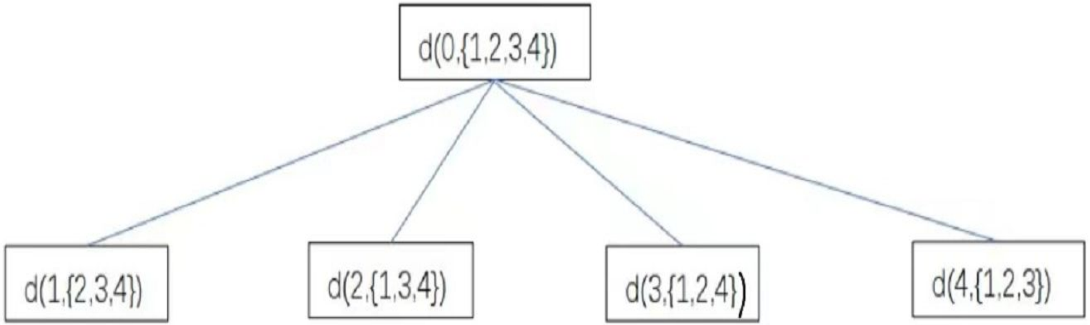
3. Bitmask and dp compression

How to represent V in the dp matrix?

Example

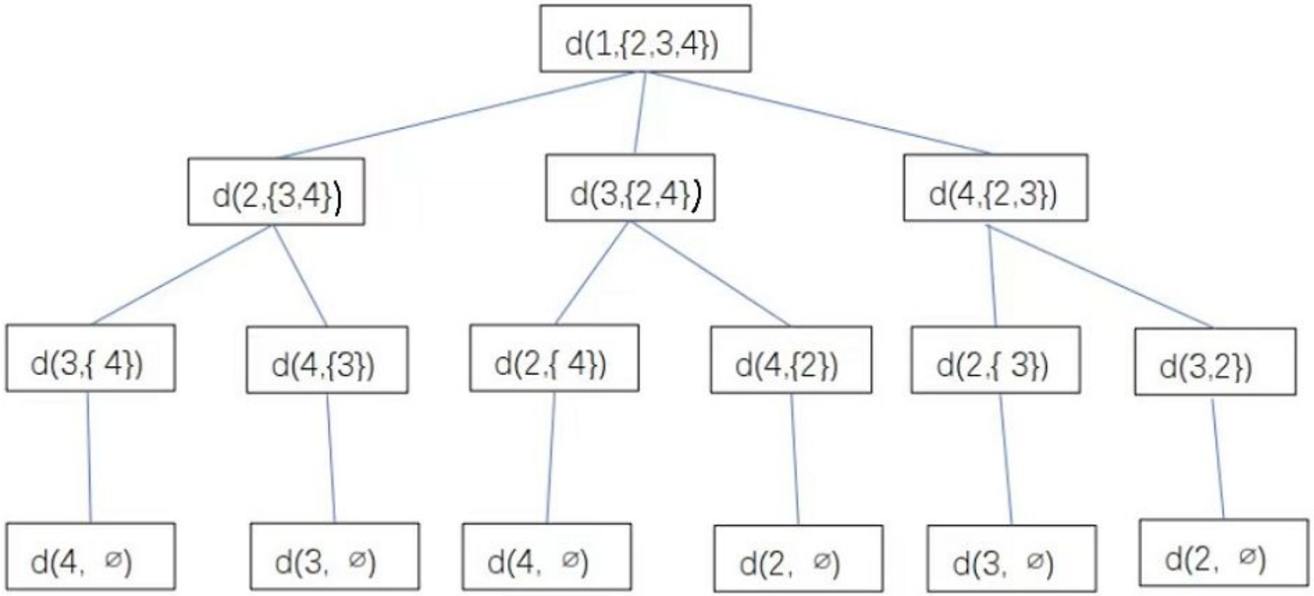
dp																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j i	{Ø}	{1}	{2}	{1,2}	{3}	{1,3}	{2,3}	{1,2,3}	{4}	{1,4}	{2,4}	{1,2,4}	{3,4}	{1,3,4}	{2,3,4,}	{1,2,3,4}
0																
1																
2																
3																
4																

Example



$$d(0,\{1,2,3,4\}) = \min \left\{ \begin{array}{l} C_{01}+d(1,\{2,3,4\}) \\ C_{02}+d(2,\{1,3,4\}) \\ C_{03}+d(3,\{1,2,4\}) \\ C_{04}+d(3,\{1,2,3\}) \end{array} \right\}$$

Example



Greedy Solution to TSP: Nearest Neighbor Heuristic

Nearest-Neighbor-Heuristic(G(V,E), s):

Start at the source s,

While (there are unvisited vertices)

from the current vertex u, go to the nearest unvisited vertex v.

Return to s.

Greedy Solution to TSP: Nearest Neighbor Heuristic

Input graph

- Use an adjacent matrix to represent the graph
- graph[i][j] = graph[j][i] = the weight of edge between vertex i and vertex j

```
# Build graph
graph = [[-1 for _ in range(N)] for _ in range(N)]
for i in range(N):
    for j in range(N):
        graph[i][j] = math.sqrt((spots[i, 0] - spots[j, 0]) ** 2 + (spots[i, 1] - spots[j, 1]) ** 2)

```

Greedy Solution to TSP: Nearest Neighbor Heuristic

```python
def tsp(graph, source):
    path = [source]
    cost = 0
    visited = [0 for _ in range(N)]
    visited[source] = 1
    curr = source
    for k in range(N - 1):
        next = -1
        next_dis = float('inf')
        for i in range(N):
            if visited[i] == 0 and graph[curr][i] < next_dis:
                next_dis = graph[curr][i]
                next = i
        path.append(next)
        visited[next] = 1
        cost += next_dis
        curr = next
    path.append(source)
    return path, cost
```

Advantage

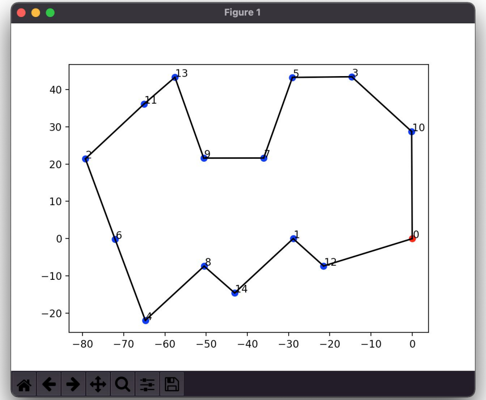
- Time Complexity: $O(V^2)$
- Fast
- Especially with large input

Disadvantage

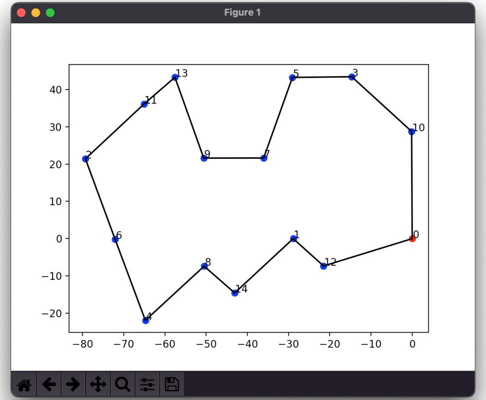
- $\Theta(\log n)$-approximation
- Does not have a constant approximate ratio
- as the number of vertices grows, the difference between the approximate solution and the optimal solution grows

Example: 15 spots

Greedy



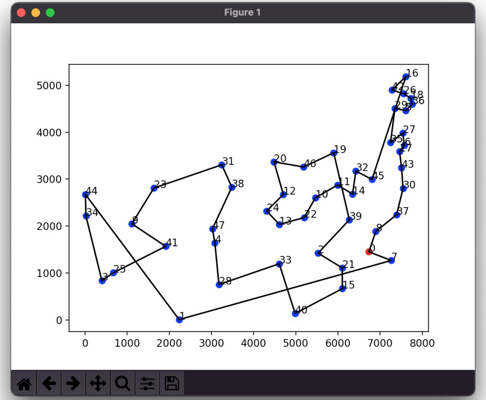
Optimal



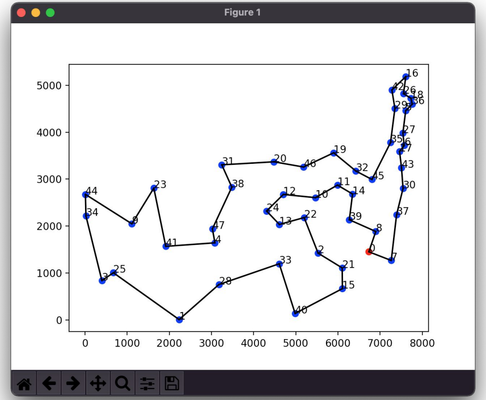
Run time = 0.002491916064172983 s

Example: 48 spots

Greedy cost = 39964



Optimal cost = 33523



Run time = 0.007245654007419944 s

Solution to TSP Problem: Christofides Algorithm

1.5-approximation, $O(V^3)$ time complexity

- inspired by 2-approximation algorithm
- designed to solve metric TSP problem (graph satisfies **Triangle Inequality**)

main idea: combine **minimum spanning tree** and **minimum weight perfect matching**

- minimum spanning tree:
  - a subset of undirected edges that connects all vertices in the graph with the minimum total weights
- minimum weight perfect matching:
  - a set of edges that covers every vertex of the graph exactly once with the minimum total weights

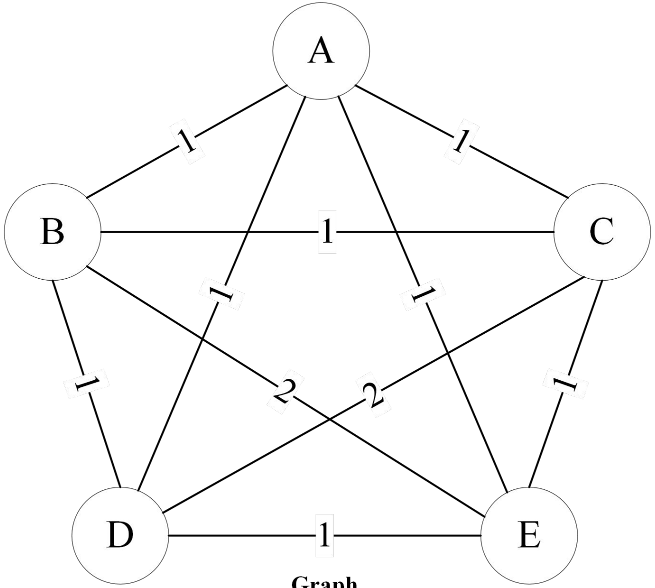
Christofides Algorithm

TSP-Christofides(Graph)

1. Create a **Minimum Spanning Tree**(MST) T
2. Find the set of vertices O with odd degree in T
3. Create a **Minimum-Weight Perfect Matching** P in induced subgraph given by vertices from O
4. Combine the edges in T and P to form a connected **multigraph** M
5. Generate the **Eulerian Circuit** E from M
6. Generate the **Hamiltonian Circuit** H from E by skipping repeated vertices except start vertex

Return H

Example



Graph

Graph: complete, undirected, positive weighted

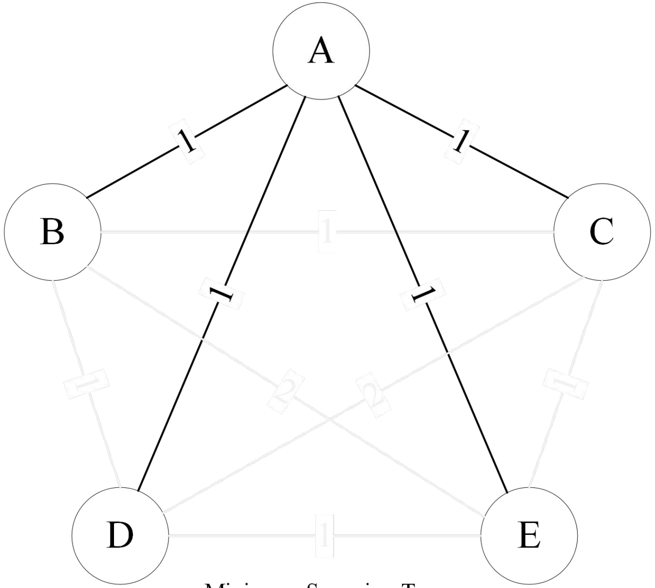
Let $d(u, v)$ denote the non-negative distance between vertex u and vertex v.

- $d(u, v) \geq 0$
- $d(u, v) = 0$ iff $u = v$
- $d(u, v) = d(v, u)$
- $d(u, v) \leq d(u, w) + d(w, v)$ -> Triangle Inequality

Let $d(A)$ denote the total distance of the edges subset A

Our goal is to find Hamiltonian Circuit with the minimum total distance.

1. Create a **Minimum Spanning Tree**(MST) T



Minimum Spanning Tree

Let H^* demote the optimal TSP tour

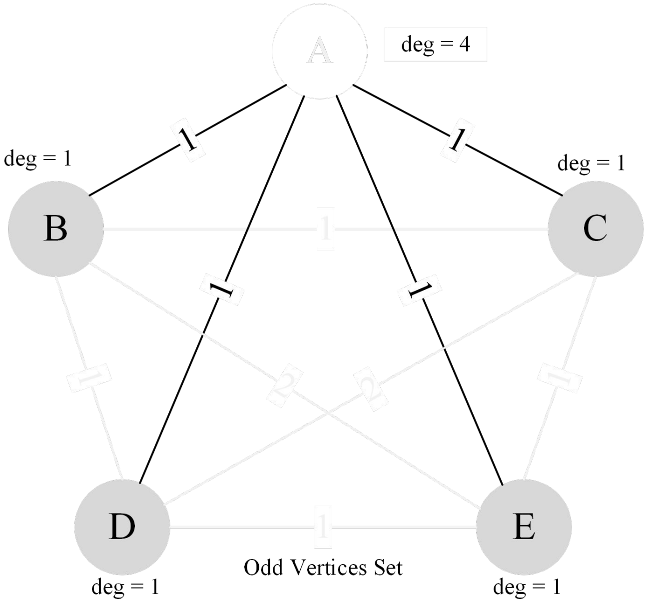
$$d(T) \leq d(H^*)$$

Time Complexity:

use Prim's algorithm

- Binary heap implementation $O(V \log V + E \log V) \to O(V^2 \log V)$
- Fibonacci heap implementation $O(E + V \log V) \to O(V^2)$

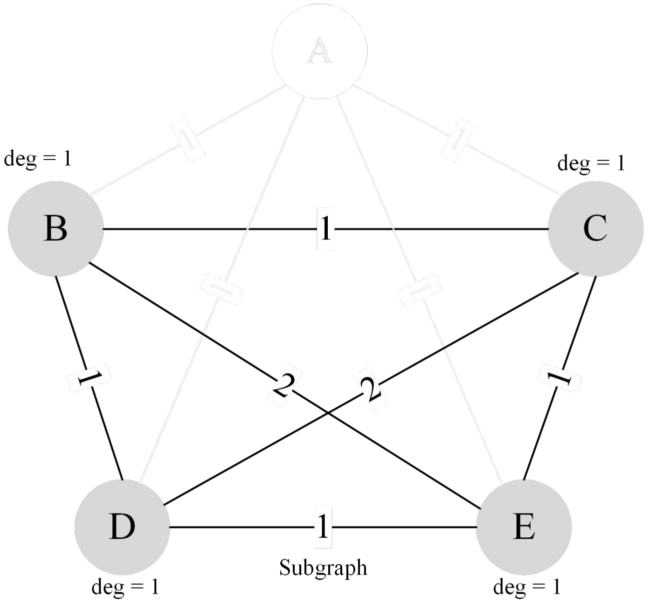
2. Find odd degree vertices set O in MST



By handshaking lemma, |O| is even.

Time Complexity: O(V)

3. Create the induced subgraph S given by vertices from O

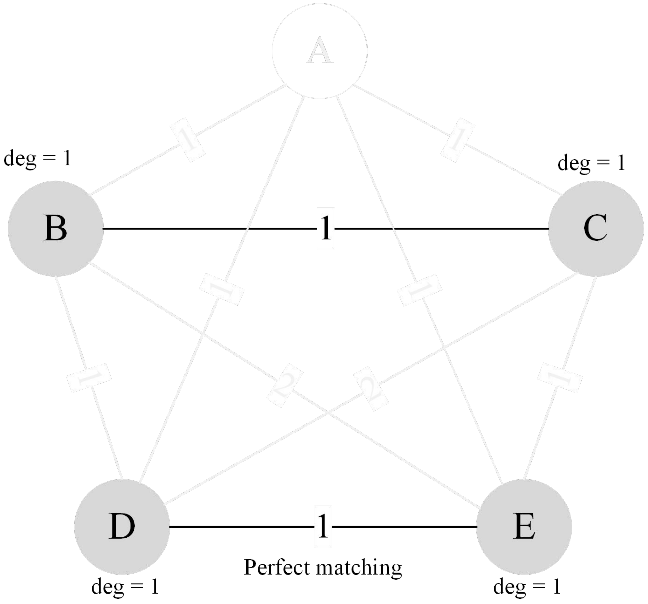


The subgraph is a complete graph.

Since |O| is even, a perfect matching must exists in this subgraph.

Time Complexity: $O(V^2)$

4. Create a **Minimum-Weight Perfect Matching** P in induced subgraph S



The number of perfect matching for a complete graph is fixed.

So we can find the one with minimum weight in polynomial-time.

Let N* demote the optimal TSP tour on subgraph, $d(N^*) \leq d(H^*)$

Let N1 and N2 be 2 alternatively selected perfect matching

$d(N1) + d(N2) = d(N^*)$

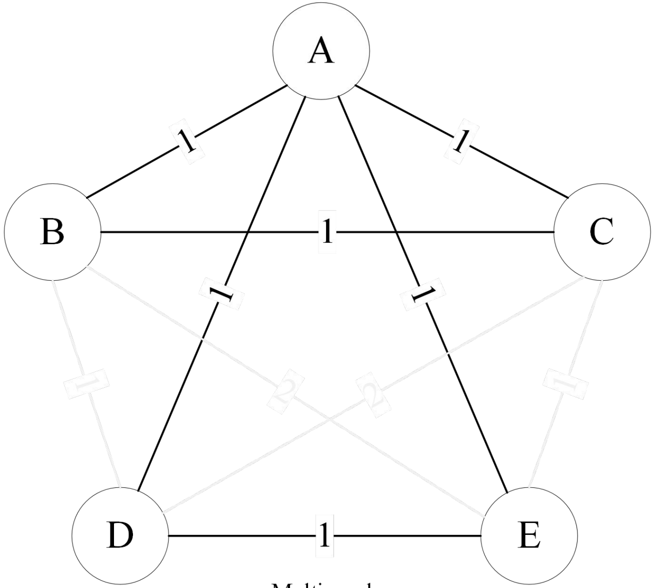
Because $d(P) \leq \min\{ d(N1), d(N2) \}$

$$d(P) \leq 1/2d(N^*) \leq 1/2d(H^*)$$

Time Complexity: $O(V^3)$

use Edmonds Blossom algorithm

5. Combine the edges in T and P to form a connected **multigraph** M



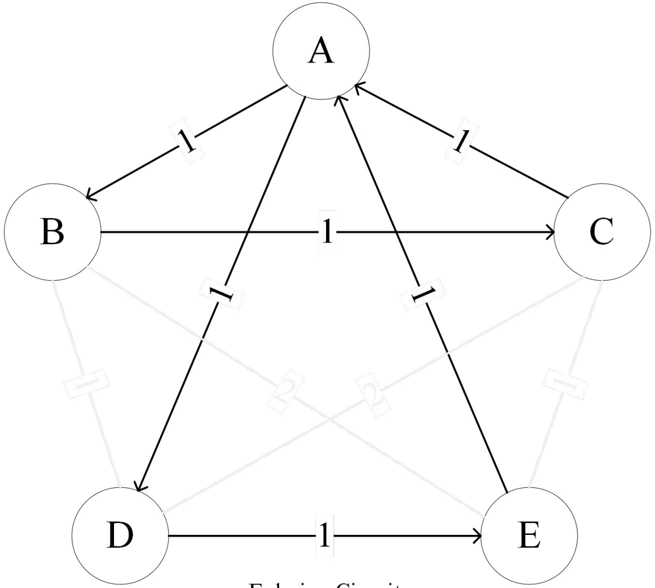
Multigraph

combine d(T) ≤ d(H*) and d(P) ≤ 1/2d(H*)

$$d(M) = d(T) + d(P) \leq 1.5d(H^*)$$

Time Complexity: O(V)

6. Generate the **Eulerian Circuit** E from M



Eulerian Circuit

Eulerian Circuit: A, B, C, A, D, E, A

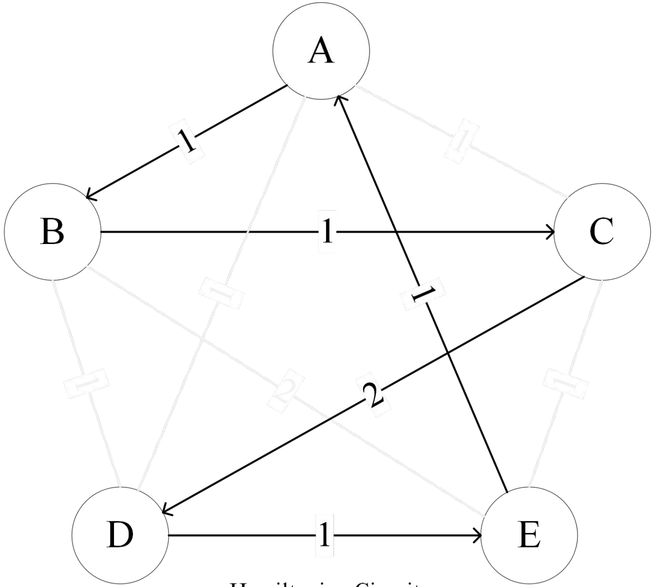
visits every edge exactly once, no changes on edge

$$d(E) = d(M) \leq 1.5d(H^*)$$

Time Complexity: O(V)

use Hierholzer's algorithm

7. Generate the **Hamiltonian Circuit** H from E by skipping repeated vertices except start vertex



Hamiltonian Circuit

Hamiltonian Circuit: A, B, C, D, E, A

delete repeated vertices except the start vertex

Since this graph follows triangle inequality, d(C, D) ≤ d(C, A) + d(A, D)

$$d(H) \le d(E) \le 1.5d(H^*)$$

Time Complexity: O(V)

Thanks

Algorithms to Solve Real Travel Planning Problems

Zhou Liu, Jifan Xie, Jing Ming

CS5800 final Project