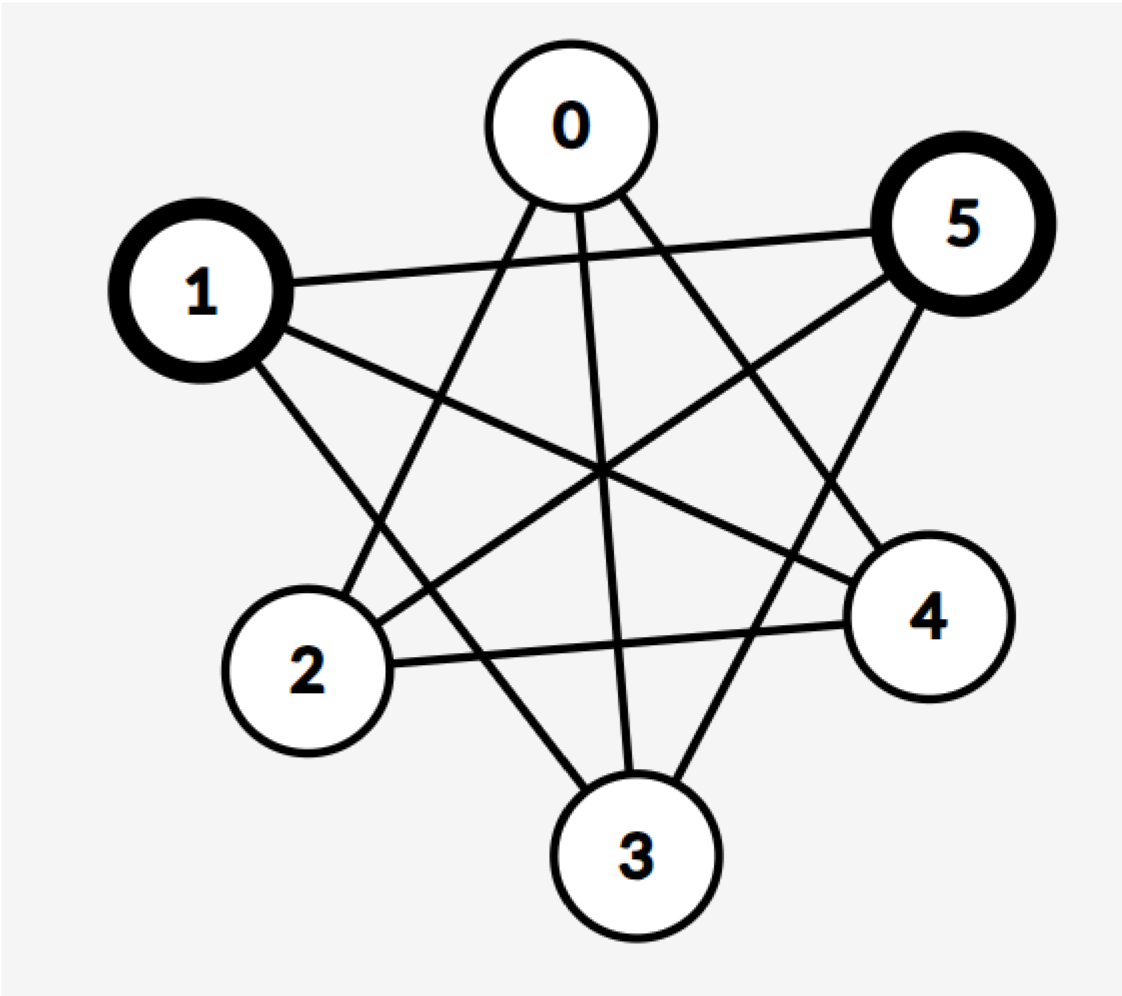
离散（2）hw4

> 王子轩 `2023011307`
>
> `wang-zx23@mails.tsinghua.edu.cn`

P13 T9

> 六个人围城圆形就坐，每个人恰好只与相邻者不认识，是否可以重新入坐，使得每个人都与邻坐认识?

解：可以。不妨画出原来的图形如图所示：



解：使用认识图上的点v表示人，v_i 和 v_j 之间相互认识的关系可以建模为 $\exists(v_i, v_j) \in E$.不妨画出原来的图形如图所示; 利用书本P31的推论2.4.2: 若简单图G的每个顶点的度都$\geq \dfrac{1}{2}n$, 则G有哈密顿回路。在本题中，有6个人，即$n = 6$。每个人恰好只与相邻者不认识，即每个人与除了左右相邻的两人外的所有人都认识。因此，在认识图中，每个顶点的度为$n - 3 = 6 - 3 = 3$。推论2.4.2的条件：$\geq \dfrac{1}{2}n = \dfrac{1}{2} \cdot 6 = 3$，每个顶点的度都等于3，恰好满足条件.该图存在哈密顿回路。给出一个哈密顿回路是$1 \to 3 \to 0 \to 4 \to 2 \to 5 \to 1$

P53 T18

> 设G是$n \geq 3$的简单图，证明：若$m \geq \frac{1}{2}(n-1)(n-2)+2$，则$G$中存在哈密顿回路。

证明：根据书本P30的推论2.4.1:若简单图G的任意v_i和v_j之间恒有$d(v_i)+d(v_j)\geq n$,则G中存在哈密顿回路。采用反证法: 我们假设G中存在v_i和v_j使得$d(v_i)+d(v_j)\leq n-1$. 对于原图G的导出子图$G'=G-\{v_i,v_j\}$而言，$G'(V',E')$的边数m'最大是当G'是完全图时候，而$|V'|=n-2$因此$m'\leq\frac{1}{2}(n-2)(n-3)$, 因此
$m\leq m'+d(v_i)+d(v_j)\leq\frac{1}{2}n^2-\frac{3}{2}n+2\leq\frac{1}{2}(n-1)(n-2)+2$

P53 T24

解：（a）图有哈密顿回路：$v_1v_7v_2v_3v_4v_5v_6v_1$

（b）图没有。（判断的方法是标注AB节点，根据二分图结论，图上节点数量为奇数因此没有）

P53 T31

> 从(0,0)出发，要求最短的经过(2,5),(9,3),(8,9),(6,6),走X, Y轴的最短行进路线

解：使用分支定界法。符号约定V_0(0,0), V_1(2,5), V_2(9,3), $V_3(8,9)$, V_4(6,6). 节点之间的距离即是曼哈顿距离。

d01	d02	d03	d04	d12	d13	d14	d23	d24	d34
7	12	17	12	9	10	5	7	6	3

按照边权值进行升序排序

d34	d14	d24	d01	d23	d12	d13	d02	d04	d03

```python
import numpy as np
coordinates = [(0, 0), (2, 5), (9, 3), (8, 9), (6, 6)]
node_names = ["V0", "V1", "V2", "V3", "V4"]
n = 5
dist_matrix = np.ones((n, n)) * float('inf')
edges = {
    (0, 1): 7, (0, 2): 12, (0, 3): 17, (0, 4): 12,
    (1, 2): 9, (1, 3): 10, (1, 4): 5,
    (2, 3): 7, (2, 4): 6,
    (3, 4): 3
}
for (i, j), dist in edges.items():
    dist_matrix[i, j] = dist
    dist_matrix[j, i] = dist
sorted_edges = sorted(edges.items(), key=lambda x: x[1])
print("边的权值升序排列:")
for (i, j), dist in sorted_edges:
    print(f"({node_names[i]}, {node_names[j]}): {dist}")
def dfs_branch_and_bound_tsp(dist_matrix, sorted_edges, start_node=3):
    n = len(dist_matrix)
    best_path = None
    best_cost = float('inf')
    step_count = 0
    print(f"\n基于边权值升序排列的DFS分支定界法搜索过程 (从{node_names[start_node]}开始):")
    print("-" * 90)
```

```
    print(f"{'步骤':^6} | {'当前路径':^30} | {'当前代价':^10} | {'下界':^10} | {'操作':^20}")
    print("-" * 90)
    adj_list = [[] for _ in range(n)]
    for (i, j), dist in sorted_edges:
        adj_list[i].append((j, dist))
        adj_list[j].append((i, dist))
    for i in range(n):
        adj_list[i].sort(key=lambda x: x[1])
    def dfs(path, cost, lower_bound):
        nonlocal best_path, best_cost, step_count
        step_count += 1
        current = path[-1]
        path_str = f"[{','.join([node_names[i] for i in path])}]"
        print(f"{step_count:^6} | {path_str:^30} | {cost:^10.1f} | {lower_bound:^10.1f} | {'扩展节点':^20}")
        if len(path) == n:
            total_cost = cost + dist_matrix[current][start_node]
            if total_cost < best_cost:
                best_cost = total_cost
                best_path = path + [start_node]
                print(f"{step_count:^6} | {path_str+'-'+node_names[start_node]:^30} | {total_cost:^10.1f} | {'-':^10} | {'更新最优解':^20}")
            else:
                print(f"{step_count:^6} | {path_str+'-'+node_names[start_node]:^30} | {total_cost:^10.1f} | {'-':^10} | {'不更新最优解':^20}")
            return
        for next_node, edge_dist in adj_list[current]:
            if next_node not in path:
                new_cost = cost + edge_dist
                lower_bound = new_cost
                if lower_bound >= best_cost:
                    new_path_str = f"[{','.join([node_names[i] for i in path + [next_node]])}]"
                    print(f"{step_count:^6} | {new_path_str:^30} | {new_cost:^10.1f} | {lower_bound:^10.1f} | {'剪枝':^20}")
                    continue
                dfs(path + [next_node], new_cost, lower_bound)
    dfs([start_node], 0, 0)

    print("-" * 90)
    return best_path, best_cost
start_node = 3
best_path, best_cost = dfs_branch_and_bound_tsp(dist_matrix, sorted_edges, start_node)

print("\n最终结果:")
print(f"最短哈密顿回路: {[node_names[i] for i in best_path]}")
print(f"最短路径长度: {best_cost}")

# 打印详细路径
path_description = "最短路径: "
for i in range(len(best_path)):
    node = best_path[i]
```

```
    path_description += f"{node_names[node]}({coordinates[node][0]},
{coordinates[node][1]})"
    if i < len(best_path) - 1:
        next_node = best_path[i+1]
        dist = dist_matrix[node][next_node]
        path_description += f" --{dist}--> "

print(path_description)
```

最短路径: V3(8,9) --3.0--> V4(6,6) --5.0--> V1(2,5) --7.0--> V0(0,0) --12.0--> V2(9,3) --7.0--> V3(8,9)

最短路径长度: 34.0