# 状态压缩$dp$

chenjiuri_zhuangyadpbasic123

- 状态压缩是什么?
  - 反正它是一种$dp$,所以就没有脱离小规模子问题解的重复利用。
  - 状态压缩是一类问题+技巧。
    - 思想：用代价最小的方式去表示一种状态。通常是一组二进制数。

---

## 经典问题

- 旅行商问题

> TSP问题：
> 给出一系列的城市以及它们之间的路线距离。
> 求解访问每一个城市，并且回到起始点的最短回路。

- 枚举角度
  - 如果是一个完全图，枚举方案数将高达$n!$.
- 动态规划的角度；
  - 在以往的问题中，非常容易地就可以表示区分表示某一些问题；
    - 背包问题，树形$dp$问题，数位$dp$。都有一些共同的特征，一些子问题之间的区别是连续的。并且和资源的一个数的属性有关。比较容易地小范围地表示。也就是说，子问题的序号是天然可表达的。
  - 对于旅行商的问题：
    - 我们关注的子问题是，已经经过了某一些点，众多方案中的最小值。
      - 点的数目角度上设置状态。有重复/
      - 一个具体的数组上表示状态，查询，以及区分上，都要花费比较大的时间代价。
    - 现在考虑，自己认为的将子问题标一个序号。
      - 当问题规模比较小时，用一个二进制串来表示状态。如果第一位上的状态是1说明当前的点与i经选了。
      - 好处
        - 当前状态下，用一些位运算可以高效的查询某个点是否存在。
      - 局限性
        - 这意味着维护的资源中，对像必须只有两种状态。

---

*code*

```
//------code-----٩(´ω`*)و ------靓仔代码-----٩(´ω`*)و ----talk is
cheap , show me the code--------
int d[maxn][maxn],f[1LL << maxn][maxn],n; //当前当前遍历的点为s，且当前
在
int get_dp(int s, int now) {                    //表示当前状态的情况，以及集合。
 //迁移的情况下，类似深度优先搜索的迁移。
//此时已经计算好了当前问题的dp
    if (f[s][now] >= 0)
        return f[s][now];
    //当前状态终止。
    if (s == (1 << n) - 1 && now == 0)
        return f[s][now] = 0;
    int res = inf;
    //向下转移状态。
    for (int u = 0; u < n; u++)
        if (s >> u & 1) //看当前下一个点有没有走过。
            res = min(res, get_dp(s | 1 << u, u) + d[now][u]);
    return f[s][now] = res;
}
void MAIN(){
    cout << get_dp(0, 0) << '\n';
}
```

## 生长思考；

- 上面问题的复杂度为多少?
  - 对于s一共有$2^n$种可能。
  - 对于每一种具体的集合形式：可能有n个当前点状态。
  - 对于当前点的状态，又有若n个的迁移。
  - 综上复杂度应该是$O(2^n \times n^2)$
- 进一步抽象为二重循环的优化。
  - 发现。从大到小的枚举s可以处可以把处理一个状态时，可以保证更小规模的状态已经解决。
  - 改写成循环。

---

## 多重循环的方式，优化当前状态。

```
#include <bits/stdc++.h>
using namespace std;

void MAIN();
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
```

```
    MAIN();
}
typedef long long ll;
const int maxn = 16;
const int inf = 0xffffff;
//------code-----٩(´ω`*)و ------靓仔代码-----٩(´ω`*)و ----talk is cheap ,
show me the code--------
ll d[maxn][maxn];
ll f[1LL << maxn][maxn]; //当前当前遍历的点为s，且当前在
int n;
void MAIN()
{
    int k;
    cin >> n >> k;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            d[i][j] = inf, f[i][j] = -1;
    for (int i = 1; i < k; i++)
    {
        int x, y, s;
        cin >> x >> y >> s;
        d[x][y] = d[y][x] = s;
    }
    f[(1 << n) - 1][0] = 0;
    for (ll s = (1 << n) - 2; s >= 0; s--) //初始的s应该是111111这样的形式。
        for (int v = 0; v < n; v++)        //如果当前v点实际上是不属于集合中的
点会怎么样发展?
        {
            if (s >> v & 1) //当前的u点有没有在集合之中。
                continue;
            for (int u = 0; u < n; u++) //将要前往的点。怎么折腾都没事，反正这
种状态的点最终是用不到的
                if (!(s >> u & 1))       //判断当前点是否已经走过。
                    f[s][v] = min(f[s][v], f[s | 1 << u][u] + d[u][v]);
        }
    cout << f[0][0] << '\n';
}
```