**matrix-chain multiplication DP: OPT(i,j)=min\_(i≤k<j) {OPT(i,k),+OPT(k+1,j)+p\_(i-1) p\_k p\_j}, optimal cost to compute A\_i...A\_j. T(n)=O(n^3 ),S=O(n^2 )**

**Bellman-Ford:一个点到其他点的最短路径。n-1迭代计算起始点到其他顶点的最短路径。迭代中依次增加能够包含的边的数量。T(n)=O(VE)**

**Floyd-Warshall:所有点到所有点的最短路径。通过中间顶点逐步更新两点的最短路径,可能有多个中间顶点subproblem，故使用DP。T(n)=O(V^3 ),S=O(V^2 )**

**QuickSort:原地算法。选择基准，基准左侧为小于基准，右侧为大于，重复过程直到有序D&C:T(n)=O(nlogn)**

**R-Quicksort:随机选择基准。T(n)=O(nlogn),比普通quicksort更稳定。**

**Hashing:使用字典。理论上插入，新增，搜索，删除操作都只需常数时间O(1)。但为应对hash collision,加入chain hashing,即对哈希表的每个位置存储链表，该表存储相同哈希值的键值对。加入随机化会使哈希值散布更均匀，减少hash collision.**

**Bloom Filter:宁杀错不放过，将多个h函数的哈希值存储在bit array中，进行对比。全中的那就在该哈希表中，若一个不中就不在。**

**matrix-chain multiplication DP: OPT(i,j)=min\_(i≤k<j) {OPT(i,k),+OPT(k+1,j)+p\_(i-1) p\_k p\_j}, optimal cost to compute A\_i...A\_j. T(n)=O(n^3 ),S=O(n^2 )**

**Bellman-Ford:一个点到其他点的最短路径。n-1迭代计算起始点到其他顶点的最短路径。迭代中依次增加能够包含的边的数量。T(n)=O(VE)**

**Floyd-Warshall:所有点到所有点的最短路径。通过中间顶点逐步更新两点的最短路径,可能有多个中间顶点subproblem，故使用DP。T(n)=O(V^3 ),S=O(V^2 )**

**QuickSort:原地算法。选择基准，基准左侧为小于基准，右侧为大于，重复过程直到有序。D&C:T(n)=O(nlogn)**

**R-Quicksort:随机选择基准。T(n)=O(nlogn),比普通quicksort更稳定。**

**Hashing:使用字典。理论上插入，新增，搜索，删除操作都只需常数时间O(1)。但为应对hash collision,加入chain hashing,即对哈希表的每个位置存储链表，该表存储相同哈希值的键值对。加入随机化会使哈希值散布更均匀，减少hash collision.**

**Bloom Filter:宁杀错不放过，将多个h函数的哈希值存储在bit array中，进行对比。全中的那就在该哈希表中，若一个不中就不在。**

**MaxFlow&MinCut:问题可以互相转换.用Ford-Fulkerson(FF)算法解决。**

**FF:创建残余图(residual graph);每次迭代都会寻找增广路径，即这条路径有残余容量，沿着这条路径增加流量;更新residual graph.T=O(V+E)**

**BipartiteMatching:同一集合内的点之间不相连。可以将二分图的最大匹配问题归约(reduction)成最大流最小割问题：增加一个super sink和super source**

**P：算法的求解器能够在多项式时间内得到解。**

**NP:算法的验证器能够在多项式时间内验证求解器得到的解是否正确**

**NP-Complete:是NP类问题，且所有NP类问题可以在多项式时间内归约成该问题**

**NP-Hard:该问题不一定是NP类问题，但至少与NP类问题中最难的哪一类一样难，同时其他NP类问题能够在多项式时间内归约成该问题。**

**SAT：布尔满足性问题。判断一个给定的布尔表达式是否存在一种变量赋值使得整个表达式为真。属于NP类。SAT问题的布尔算式都是CNF,即括号中用OR连接，括号之间用AND连接**

**Circuit-SAT:判断一个布尔电路是否存在一种输入使得输出为真**

**MaxFlow&MinCut:问题可以互相转换.用Ford-Fulkerson(FF)算法解决。**

**FF:创建残余图(residual graph);每次迭代都会寻找增广路径，即这条路径有残余容量，沿着这条路径增加流量;更新residual graph.T=O(V+E)**

**BipartiteMatching:同一集合内的点之间不相连。可以将二分图的最大匹配问题归约(reduction)成最大流最小割问题：增加一个super sink和super source**

**P：算法的求解器能够在多项式时间内得到解。**

**NP:算法的验证器能够在多项式时间内验证求解器得到的解是否正确**

**NP-Complete:是NP类问题，且所有NP类问题可以在多项式时间内归约成该问题**

**NP-Hard:该问题不一定是NP类问题，但至少与NP类问题中最难的哪一类一样难，同时其他NP类问题能够在多项式时间内归约成该问题。**

**SAT：布尔满足性问题。判断一个给定的布尔表达式是否存在一种变量赋值使得整个表达式为真。属于NP类。SAT问题的布尔算式都是CNF,即括号中用OR连接，括号之间用AND连接**

**Circuit-SAT:判断一个布尔电路是否存在一种输入使得输出为真**

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

**Simple path** for distinct nodes; **Simple cycle** for distinct paths; **Strongly Connect Components(SCC)** for bidirectional node.

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**WaterPouringBFS**: two bottles with capacity X and Y with initial water x and y. Find possible or not that A liters water should in any bottle. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

**Simple path** for distinct nodes; **Simple cycle** for distinct paths; **Strongly Connect Components(SCC)** for bidirectional node.

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**WaterPouringBFS**: two bottles with capacity X and Y with initial water x and y. Find possible or not that A liters water should in any bottle. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

**Simple path** for distinct nodes; **Simple cycle** for distinct paths; **Strongly Connect Components(SCC)** for bidirectional node.

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**WaterPouringBFS**: two bottles with capacity X and Y with initial water x and y. Find possible or not that A liters water should in any bottle. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)

**Upper**: Big O , **Lower**: Big , **Tight**: Big , little o , little w

**Simple path** for distinct nodes; **Simple cycle** for distinct paths; **Strongly Connect Components(SCC)** for bidirectional node.

**Recursive Fibonacci**:

**Non-recursive Fib**:

**Fib mul add**:

**BFS\_CutNode**: find node v between s and t which will destroy all s-t path if v is deleted. ->

**Graph\_isOdd**: find if a directed graph G has an odd-length cycle. ->

**WaterPouringBFS**: two bottles with capacity X and Y with initial water x and y. Find possible or not that A liters water should in any bottle. ->

**Path\_Num**: Find the number of path of two nodes in a graph: use topo\_order() find topology order of the node, then use two for loops to sum each topo order elements’ neighbor to output the path number. -> T(n) = O(V+E)