GeekBand 极客班

互联网人才 + 油站!

GeekBand 极客班

www.geekband.com

7. Graph & Search

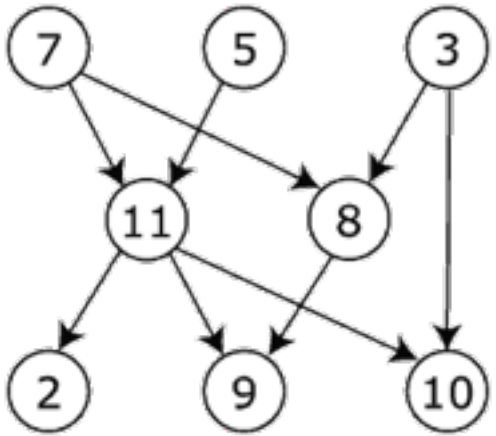
大纲

1. 图介绍和表示方式
2. 宽度优先搜索BFS
3. 深度优先搜索DFS
4. 排列组合问题
5. 回溯算法，图论面试题实战

图介绍

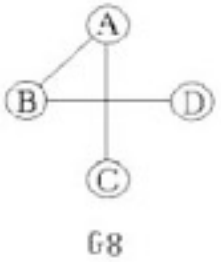
图是节点集合的一个拓扑结构，节点之间通过边相连。图分为有向图和无向图。有向图的边具有指向性，即A－B仅表示由A到B的路径，但并不意味着B可以连到A。与之对应地，无向图的每条边都表示一条双向路径。

DAG Directed acyclic graph

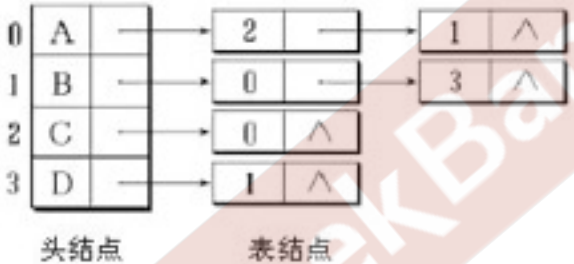


表示方式

图的数据表示方式也分为两种，即邻接表(adjacency list)和邻接矩阵(adjacency matrix)。

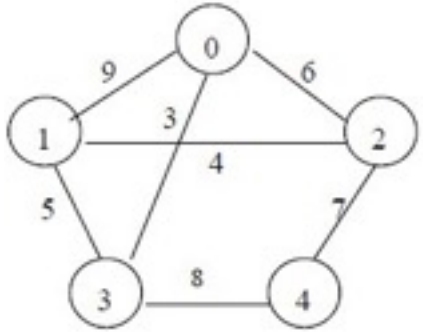


G8



$$A=\begin{pmatrix}0&1&0&1\\1&0&1&1\\0&1&0&0\\1&1&0&0\end{pmatrix}$$

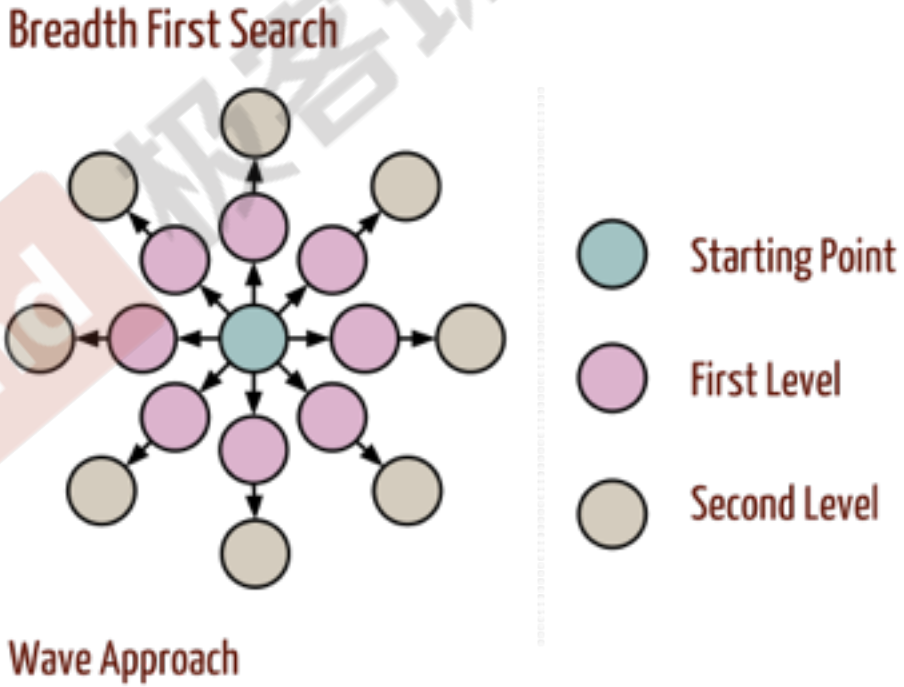
图 8.7　一个无向图的邻接矩阵表示



$$A=\begin{pmatrix}\infty&9&6&3&\infty\\9&\infty&4&5&\infty\\6&4&\infty&\infty&7\\3&5&\infty&\infty&8\\\infty&\infty&7&8&\infty\end{pmatrix}$$

宽度优先

```
BFS(G, s)
For each vertex u exept s
        Do Color[u] = WHITE
           Distance[u] = MAX
           Parent[u] = NIL
Color[s] = GRAY
Distance[s] = 0
Parent[s] = NIL
Enqueue(Q, s)
While Q not empty
        Do u = Dequeue(Q)
           For each v is the neighbor of u
               Do if Color[v] == WHITE
                  Color[v] = GRAY
                  Distance[v] = Distance[u] + 1
                  Parent[v] = u
                  Enqueue(Q, v)
            Color[u] = BLACK
```



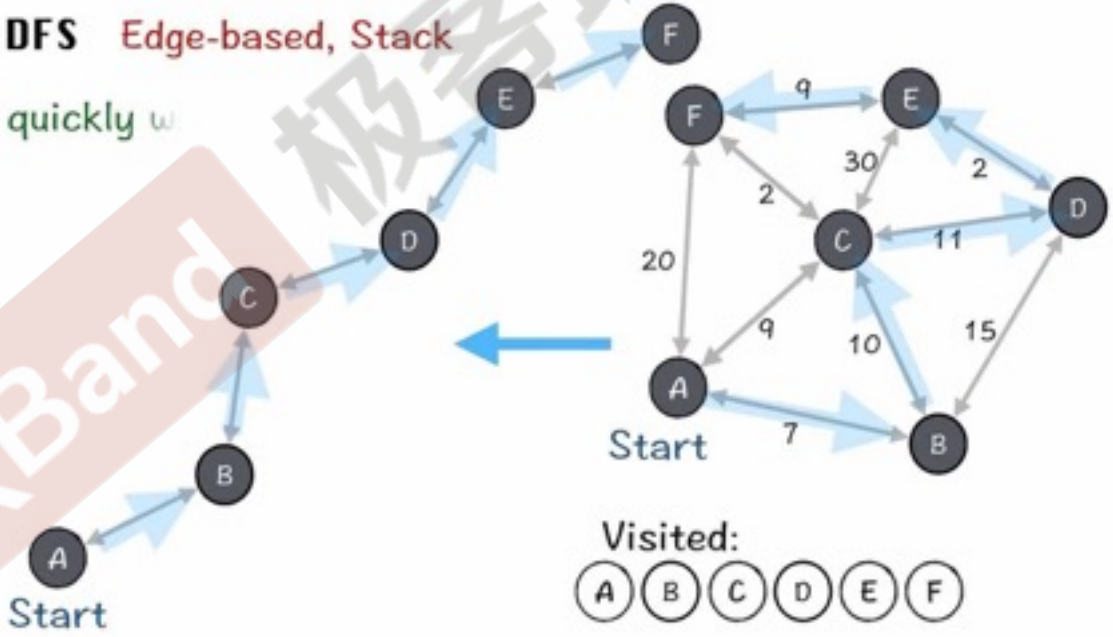
BFS特点

搜索所有可以到达的状态，转移顺序为『初始状态->只需一次转移就可到达的所有状态->只需两次转移就可到达的所有状态->...』，所以对于同一个状态，BFS 只搜索一次

BFS 通常配合队列一起使用，搜索时先将状态加入到队列中，然后从队列顶端不断取出状态，再把从该状态可转移到的状态中尚未访问过的部分加入队列，知道队列为空或已找到解。因此 BFS 适合用于『由近及远』的搜索，比较适合用于求解最短路径、最少操作之类的问题。

深度优先

```
DFS(G)
For each vertex v in G
    Do Color[v] = WHITE
    Parent[v] = NIL
For each vertex v in G
    DFS_Visit(v)

DFS_Visit(u)
Color[u] = GRAY
For each v is the neighbor of u
    If Color[v] == WHITE
        Parent[v] = u
        DFS_Visit(v)
Color[u] = BLACK
```



单源最短路径问题

对于每条边都有一个权值的图来说，单源最短路径问题是指从某个节点出发，到其他节点的最短距离。该问题的常见算法有Bellman-Ford和Dijkstra算法。

如果每条边权值相同(无权图)，由于从源开始访问图遇到节点的最小深度就等于到该节点的最短路径，因此 Priority Queue就退化成Queue，Dijkstra算法就退化成BFS。

Dijkstra 算法

```
DIJKSTRA(G, s)
S = EMPTY
Insert all vertexes into Q
While Q is not empty
    u = Q.top
    S.insert(u)
    For each v is the neighbor of u
        If d[v] > d[u] + weight(u, v)
            d[v] = d[u] + weight(u, v)
            parent[v] = u
```

All-pairs shortest paths

FLOYD(G)
$Distance^{(0)}$ = Weight(G)
For k = 1 to n
    For i = 1 to n
        For j = 1 to n
      $Distance^{(k)}_{ij}$ = min($Distance^{(k-1)}_{ij}$, $Distance^{(k-1)}_{ik}$ + $Distance^{(k-1)}_{kj}$)
Return $Distance^{(n)}$

回溯

DFS 通常从某个状态开始，根据特定的规则转移状态，直至无法转移(节点为空)，然后回退到之前一步状态，继续按照指定规则转移状态，直至遍历完所有状态。

回溯法包含了多类问题，模板类似。
排列组合模板->搜索问题(是否要排序，哪些情况要跳过)

使用回溯法的一般步骤：
确定所给问题的解空间：首先应明确定义问题的解空间，解空间中至少包含问题的一个解。
确定结点的扩展搜索规则
以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

模式识别

用Backtracking(Top-Down)解决发散结构问题

对于发散性问题(例如“所有组合”，“全部解”)，可以选取其问题空间“收敛”的一端作为起点，沿着节点发散的方向(或者说，当前节点的多种选择)进行递归，直到a.当前节点“不合法” 或 b.当前节点发散方向搜索完毕，才会return。

如果需要记录决策的路径，可以用vector<int> &path沿着搜索的方向记录，在满足胜利条件时记录当前path(通常是将path存入vector<vector<int>> &paths)。

Backtracking的典型模板c

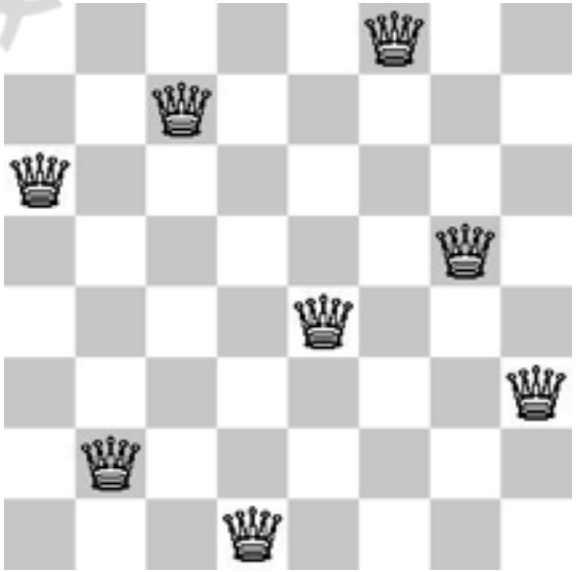
```
void backtracking( P node, vector<P> &path, vector<vector<P> >&paths ){
        if(!node )  // invalid node
            return;

        path.push_back(node);

        bool success =  ;  // condition for success
        if( success )
            paths.push_back( vector<P>(path.begin(),path.end()) ); // don't return here

        for( P next: all directions )
            backtracking( next, path, paths );
        path.pop_back();
        return;
}
```

Parentheses

Given n pairs of parentheses, generate all valid combinations of parentheses. E.g. if n = 2, you should return ()(), (())

N Queen

Please write a function to find all ways to place n queens on an n×n chessboard such that no two queens attack each other.



部分排列

问题：从0-9这十个数字中选取3个数字进行排列，打印所有的组合结果。

全排列是求出了所有字母的排列方式，该题是求出了部分字母排列的方法。只需要将结束的条件由if (index == size - 1)改为 if (index == m)即可，其中m是指要进行全排列的字符个数

```
//从size个字符中选取m个字符进行全排列，打印排列结果
void Permutation(char str[], int index, int m, int size) {
    if (index == m) {  //和本文第一个程序比只是这里打印时不太相同
        for (int i = 0; i < m; i++)
            cout << str[i];
        cout << "\t";
    }
    else {
        for (int i = index; i < size; i++) {
            swap(str[index], str[i]);
            Permutation(str, index + 1, size);
            swap(str[index], str[i]);
        }
    }
}
```

Subsets

Given a set of distinct integers, return all possible subsets.

Note
Elements in a subset must be in non-descending order.

The solution set must not contain duplicate subsets.

Example
If S = [1,2,3], a solution is:

```
[
  [3], [1], [2],[1,2,3], [1,3],[2,3], [1,2], []
]
```

使用位运算求组合问题

根据集合论知识，我们知道一个 包含N个元素的集合对应的所有子集个数为2^N。举个例子，S=1,2，那么它的全子 集=∅,{1},{2},{1,2}

子集的构造过程实际上是每个元素选与不选所构成的情况的汇总，因此 可以对每个元素赋予一个0-1数值，将所有元素的选与不选与0-1对应起来，从而就可以得到二进制数据，

[0, 2^N−1]的二进制表示。

00 -> ∅
01 -> {1}
10 -> {2}
11 -> {1,2}

```
void Combination(char *str) {
    int len=strlen(str);
    ////遍历所有的情况，1<<len就等于2^len，遍历1 ~ 2^len-1
    for(int cur=1; cur < (1<<len); cur++)  {
        for(int j=0; j < len; j++)  //遍历所有的字符 {
                if(cur & (1 << j))  //判断哪一位为1，即输出该位
                    cout << str[j];
                }
         cout << endl;  //一种情况结束
    }
}
```

唯一性的集合

Given a collection of integers that might contain duplicates, S, return all possible subsets.
Note:
Elements in a subset must be in non-descending order.
The solution set must not contain duplicate subsets.

For example,
If S = [1,2,2], a solution is:
[
  [2], [1], [1,2,2], [2,2], [1,2], []
]

唯一性的集合

与Subsets有关，先背下Subsets的模板
既然要求Unique的，就想办法排除掉重复的。

思考哪些情况会重复？如{1, 2(1), 2(2), 2(3)}，规定{1, 2(1)}和{1, 2(2)}重复，{1, 2(1), 2(2)}和{1, 2(2), 2(3)}重复。观察规律。
得出规律：我们只关心取多少个2，不关心取哪几个。
规定必须从第一个2开始连续取（作为重复集合中的代表），如必须是{1, 2(1)}不能是{1, 2{2}}

1. 一定要先排序
2. 调用下一层递归时(ln 17)，起始index 一定是i+1而不能错写成start+1

唯一性的集合

```
1  class Solution {
2  public:
3      vector<vector<int> > subsetsWithDup(vector<int> &S) {
4          vector<vector<int>> allSets;
5          vector<int> sol;
6          allSets.push_back(sol);
7          sort(S.begin(), S.end());
8          findSubsetsWithDup(S, 0, sol, allSets);
9          return allSets;
10     }
11
12     void findSubsetsWithDup(vector<int> &S, int start, vector<int> &sol, vector<vec
13         for(int i=start; i<S.size(); i++) {
14             if(i>start && S[i]==S[i-1]) continue;
15             sol.push_back(S[i]);
16             allSets.push_back(sol);
17             findSubsetsWithDup(S, i+1, sol, allSets);
18             sol.pop_back();
19         }
20     }
21 };
```

有重复的全排列

a, b, b

b, a, b

b, b, a

1．全排列就是从第一个数字起每个数分别与它后面的数字交换。

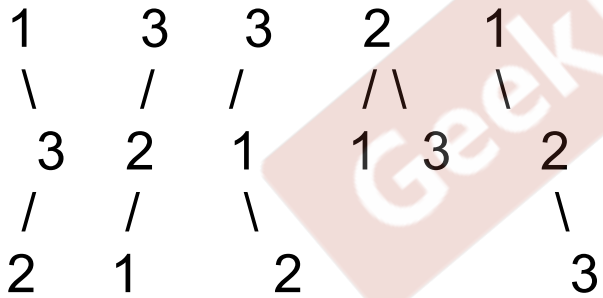
2．去重的全排列就是从第一个数字起每个数分别与它后面非重复出现的数字交换。

3．全排列的非递归就是由后向前找替换数和替换点，然后由后向前找第一个比替换数大的数与替换数交换，最后颠倒替换点后的所有数据。

Unique Binary Search Trees

Given n, generate all structurally unique BST's (binary search trees) that store values 1...n.

Example
Given n = 3, your program should return all 5 unique BST's shown below.



Combination Sum

Given two integers n and k, return all possible combinations of k numbers out of 1,2,...,n.

Combination Sum I

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.
Note:

All numbers (including target) will be positive integers.
Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
The solution set must not contain duplicate combinations.

Combination Sum II

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.
Note:

All numbers (including target) will be positive integers.

Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).

The solution set must not contain duplicate combinations.

Phone Number

Given a digit string, return all possible letter combinations that the number could represent. A mapping of digit to letters (just like on the telephone buttons) is given as below:

Input:Digit string "23"
Output:
["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

Word Ladder

Given two words (start and end), and a dictionary of words, find the length of shortest sequences of words from start to end, such that at each step only one letter is changed.
e.g: start word: hat stop word: dog
dictionary{cat, dot, cot, fat}
sequence: hat->cat->cot->dot->dog

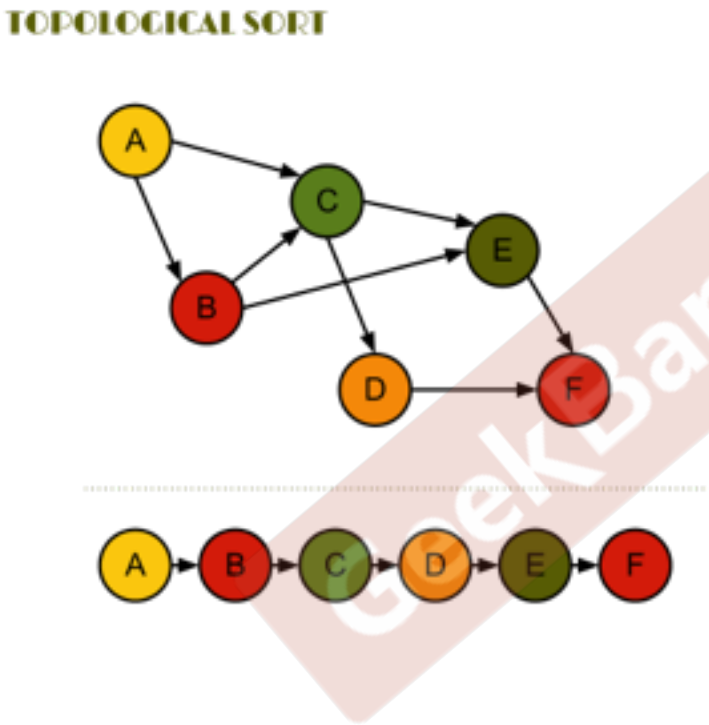
Palindrome Partitioning

Given a string s, partition s such that every substring of the partition is a palindrome. Return all possible palindrome partitioning of s. For example, given s = "aab", Return [["aa","b"], ["a","a","b"]]

图论面试题:

Clone Graph

拓扑排序



Conclusion

DFS (O(2^n), O(n!))

1. Find all possible solutions
2. Permutations / Subsets

BFS (O(m), O(n))

1. Graph traversal (每个点只遍历一次)
2. Find shortest path in a simple graph

Homework

Solve a boggle game(http://en.wikipedia.org/wiki/Boggle), with a dictionary given as unordered_set.

E	E	C	A
A	L	E	P
H	N	B	O
Q	T	T	Y

枚举n个数的所有k组合

1 1 1 0 0 //1,2,3
1 1 0 1 0 //1,2,4
1 0 1 1 0 //1,3,4
0 1 1 1 0 //2,3,4
1 1 0 0 1 //1,2,5
1 0 1 0 1 //1,3,5
0 1 1 0 1 //2,3,5
1 0 0 1 1 //1,4,5
0 1 0 1 1 //2,4,5
0 0 1 1 1 //3,4,5