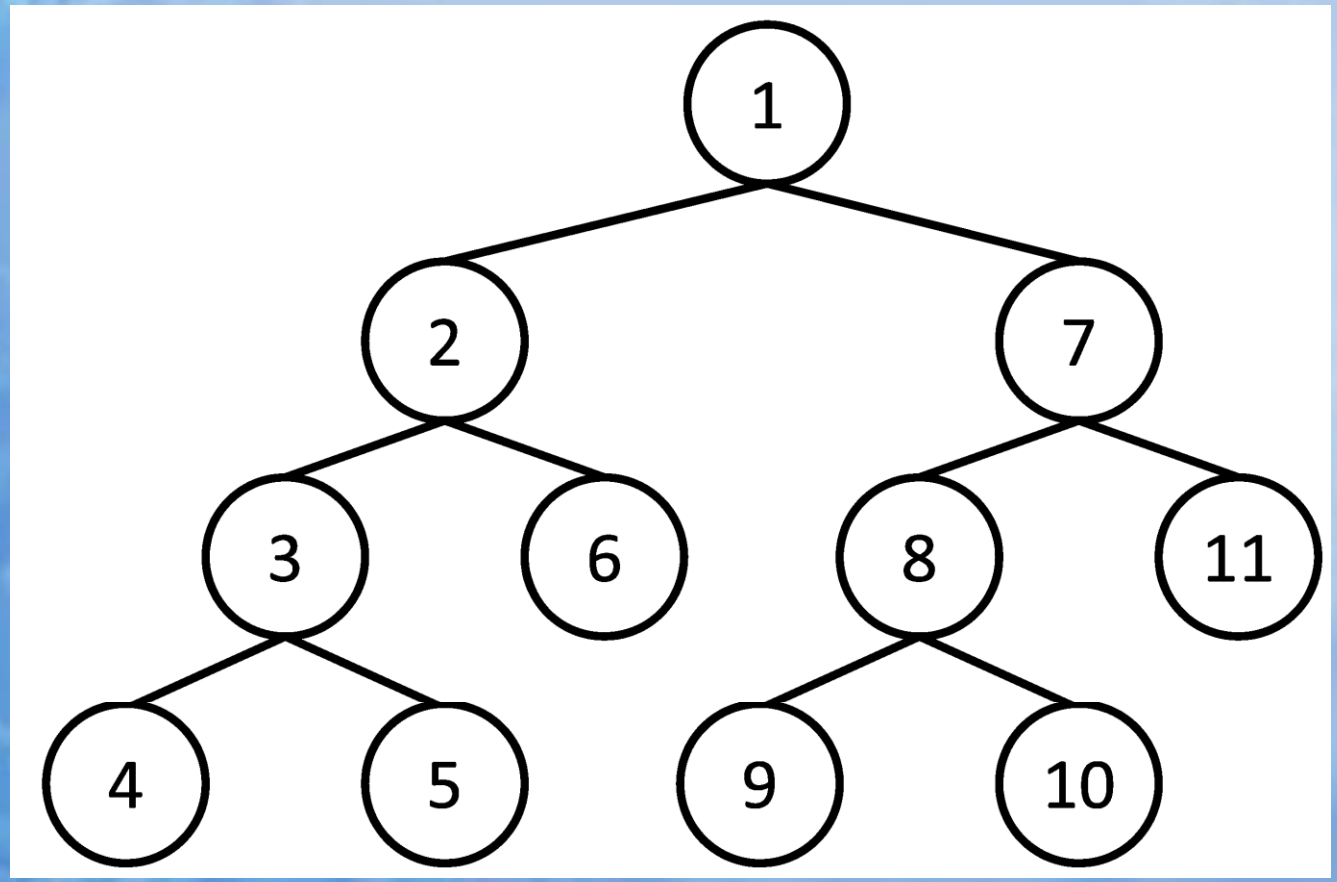
搜索

dfs

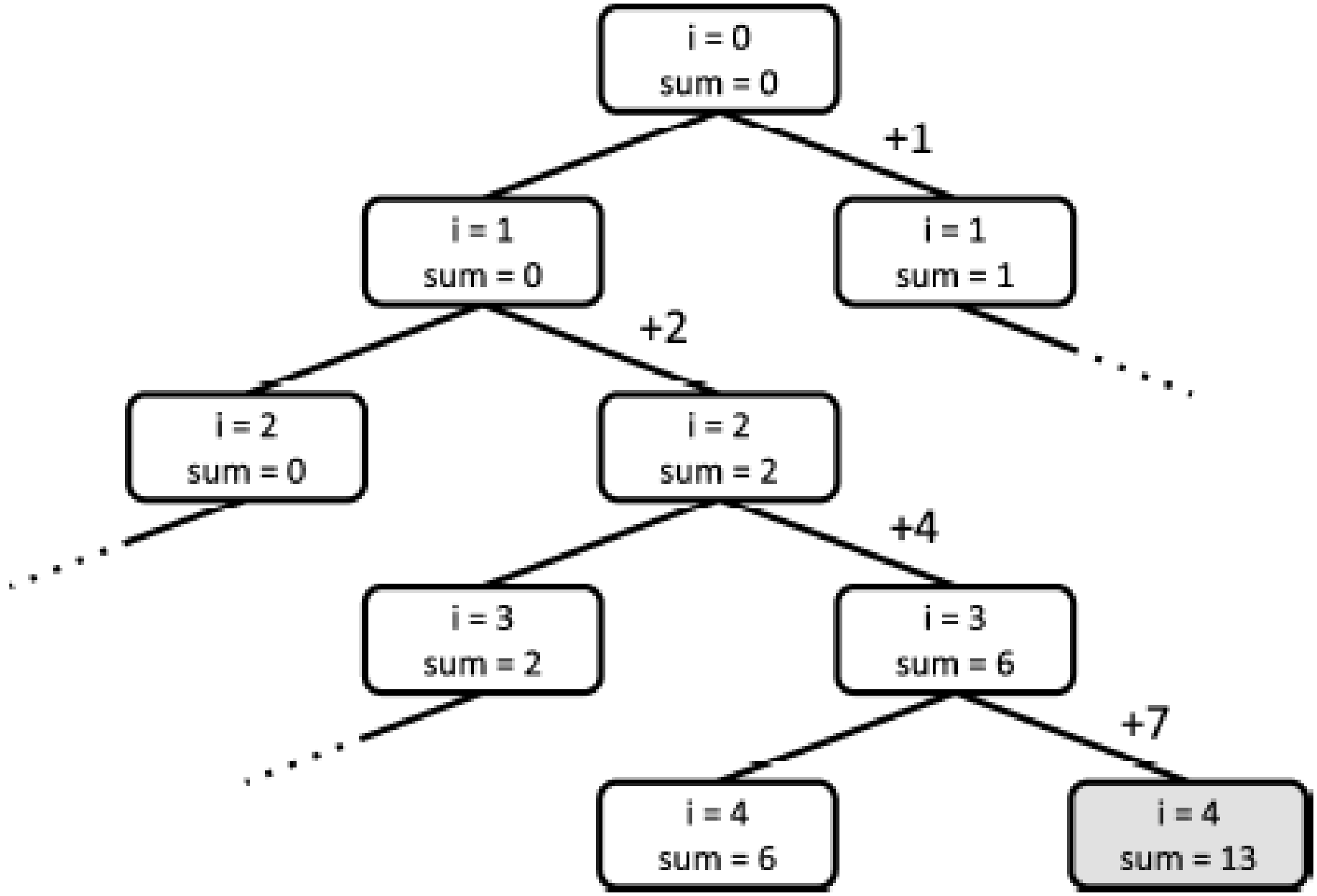
- 深度优先搜索（DFS，Depth-First Search）是搜索的手段之一。它从某个状态开始，不断地转移状态直到无法转移，然后回退到前一步的状态，继续转移到其他状态，如此不断重复，直至找到最终的解。例如求解数独，首先在某个格子内填入适当的数字，然后再继续在下一个格子内填入数字，如此继续下去。如果发现某个格子无解了，就放弃前一个格子上选择的数字，改用其他可行的数字。根据深度优先搜索的特点，采用递归函数实现比较简单

dfs



部分和问题

- 给定整数*a*1、*a*2、…、*an*，判断是否可以从中选出若干数，使它们的和恰好为*k*。
- $1 \le n \le 20$
  - $-10^8 \le a[i] \le 10^8$
  - $-10^8 \le k \le 10^8$
- 输入
- n=4, a={1, 2, 4, 7} , k=13
- n=4,a={1,2,4,7},k=15
- 输出
- `Yes (13 = 2 + 4 + 7)`
- No


i = 0
sum = 0
+1
i = 1
sum = 0
i = 1
sum = 1
+2
i = 2
sum = 0
i = 2
sum = 2
+4
i = 3
sum = 2
i = 3
sum = 6
+7
i = 4
sum = 6
i = 4
sum = 13

状态转移的样子

部分和问题

- 从*a*1开始按顺序决定每个数加或不加，在全部*n*个数都决定后再判断它们的和是不是*k*即可。因为状态数是2^*n*+1，所以复杂度是*O*(2^*n*)。如何实现这个搜索，请参见下面的代码。注意*a*的下标与题目描述中的下标偏移了1。在程序中使用的是0起始的下标规则，题目描述中则是1开始的，这一点要注意避免搞混。

部分和问题

```
int a[MAX_N], n, k;
// 已经从前i项得到了和sum，然后对于i项之后的进行分支
bool dfs(int i, int sum) {// 如果前n项都计算过了，则返回sum是否与k相等
if (i == n) return sum == k;// 不加上a[i]的情况
if (dfs(i + 1, sum)) return true;// 加上a[i]的情况
if (dfs(i + 1, sum + a[i])) return true;
// 无论是否加上a[i]都不能凑成k就返回false
return false;
}
```

HDU 3448 Bag Problem

- 给k个数，问最多取n个，所取的数的和不大于m的最大的和
- n<=40
- k<=40
- 1<= 其他 <=1000000000

HDU 3448 Bag Problem

```
void dfs(int s, int N, int M) {
    Max = max(Max, M);
    if (s >= k || N > n)return;
    dfs(s + 1, N, M);
    if (M + a[s] <= m && N + 1 <= n) {
        dfs(s + 1, N + 1, M + a[s]);
    }
}
```

Lake Counting （POJ No.2386)

有一个大小为N×M的园子，雨后积起了水。八连通的积水被认为是连接在一起的。请求出园子里总共有多少水洼？（八连通指的是下图中相对W 的*的部分）

```
***
*W*
***
```

限制条件 N, M ≤ 100

样例

输入

```
N=10, M=12
园子如下图（'W'表示积水，'.'表示没有积水）
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.
```

输出

```
3
```

Lake Counting

```
void dfs(int x,int y){
    a[x][y] = '.';
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            int nx = x + dx, ny = y + dy;
            if (0 <= nx && nx < n && 0 <= ny && ny <
m && a[nx][ny] == 'W')
                dfs(nx, ny);
        }
    }
}
```

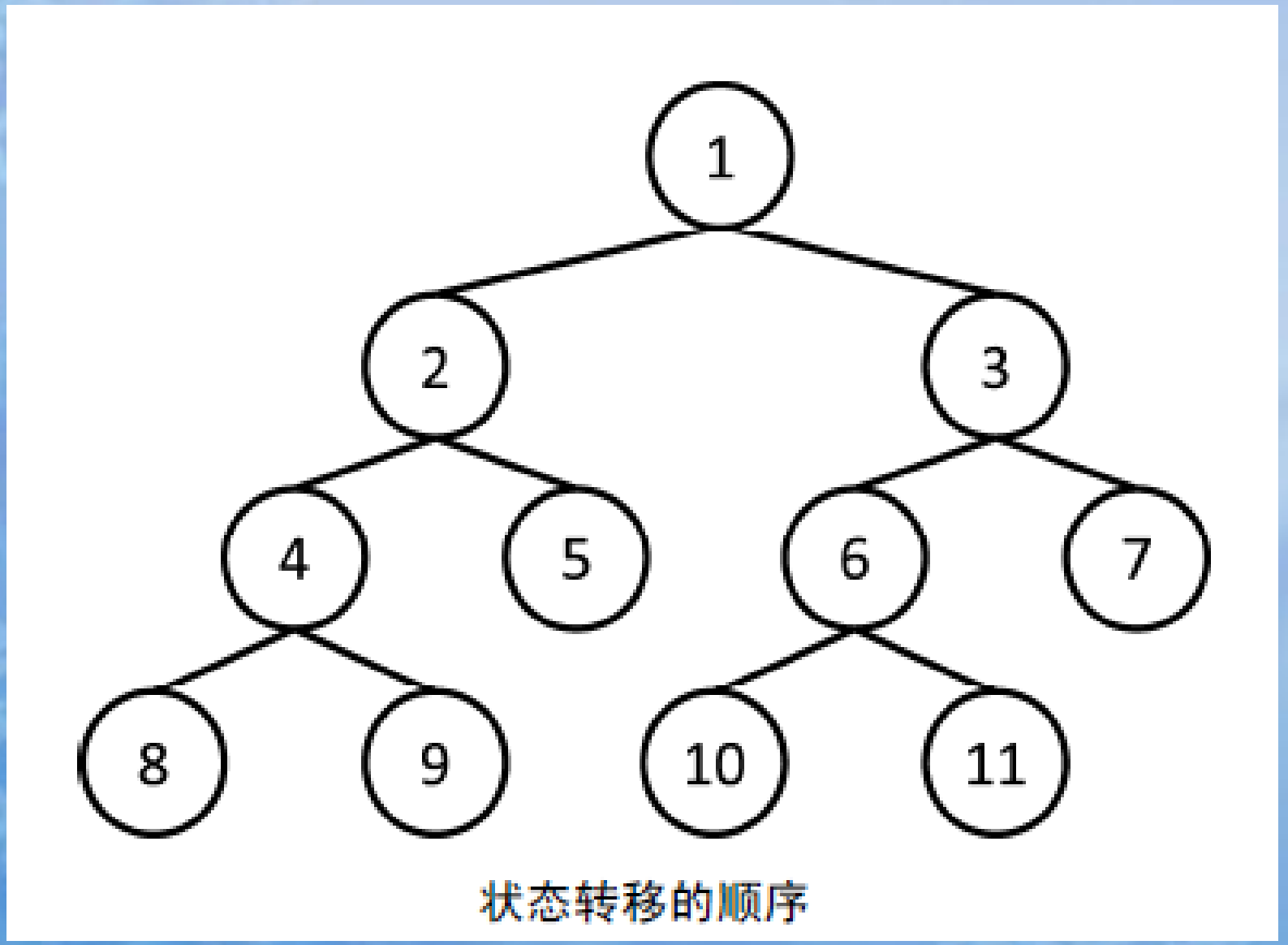
queue

```
queue<int> q;
q.push(1);
q.push(2);
q.front();//1
q.pop();
q.front();//2
```

bfs

- 宽度优先搜索（BFS，Breadth-First Search）也是搜索的手段之一。它与深度优先搜索类似，从某个状态出发探索所有可以到达的状态。
- 与深度优先搜索的不同之处在于搜索的顺序，宽度优先搜索总是先搜索距离初始状态近的状态。
- 也就是说，它是按照开始状态→只需1次转移就可以到达的所有状态→只需2次转移就可以到达的所有状态→……这样的顺序进行搜索。对于同一个状态，宽度优先搜索只经过一次，因此复杂度为*O*(状态数×转移的方式)。

bfs



状态转移的顺序

FZU – 2285 迷宫寻宝

- 洪尼玛今天准备去寻宝，在一个n*n （n行, n列）的迷宫中，存在着一个入口、一些墙壁以及一个宝藏。由于迷宫是四连通的，即在迷宫中的一个位置，只能走到与它直接相邻的其他四个位置（上、下、左、右）。现洪尼玛在迷宫的入口处，问他最少需要走几步才能拿到宝藏？若永远无法拿到宝藏，则输出-1。
- 每组数据输入第一行为正整数n，表示迷宫大小。
- 接下来n行，每行包括n个字符，其中字符'.'表示该位置为空地，字符'#'表示该位置为墙壁，字符'S'表示该位置为入口，字符'E'表示该位置为宝藏，输入数据中只有这四种字符，并且'S'和'E'仅出现一次。
- n≤1000

FZU – 2285 迷宫寻宝

›S .# . .

›# .# .#

›# .# .#

›# . . .E

›# . . . .

FZU – 2285 迷宫寻宝

```
void bfs(int xx, int yy) {
    queue<pip > q;
    q.push(make_pair(0, make_pair(xx, yy)));
    while (!q.empty()) {
        pip n = q.front();
        q.pop();
#define X (n.second.first)
#define Y (n.second.second)
#define K ( n.first)
        if (mp[X][Y] == '#')continue;
        if (mp[X][Y] == 'E') {
            ans = K;
            return;
        }
```

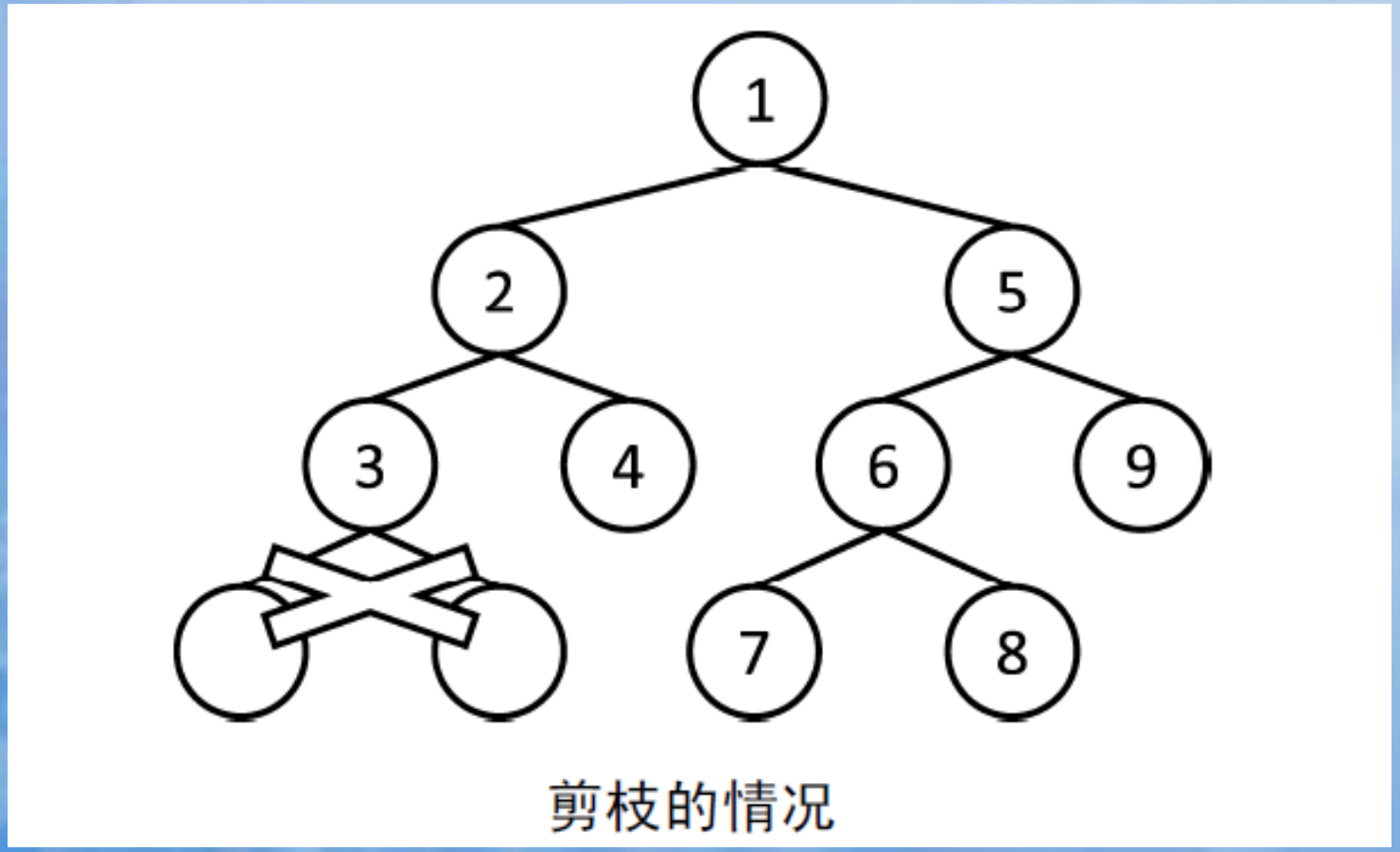
FZU – 2285 迷宫寻宝

```
 mp[X][Y] = '#';
        if (X > 0)q.push(make_pair(K + 1, make_pair(X - 1,
Y)));
        if (X < (N - 1))q.push(make_pair(K + 1,
make_pair(X + 1, Y)));
        if (Y > 0)q.push(make_pair(K + 1, make_pair(X, Y -
1)));
        if (Y < N - 1)q.push(make_pair(K + 1, make_pair(X,
Y + 1)));
    }
}
```

剪枝

- 深度优先搜索时，有时早已很明确地知道从当前状态无论如何转移都不会存在解。这种情况下，不再继续搜索而是直接跳过，这一方法被称作剪枝。
- 我们回想一下深度优先搜索的例题“部分和问题”。这个问题中的限制条件如果变为$0 \leq a_i \leq 10^8$，那么在递归中只要sum超过k了，此后无论选择哪些数都不可能让sum等于k，所以此后没有必要继续搜索。

剪枝



剪枝的情况

HDU 1010 Tempter of the Bone

- 给你一个N*M的迷宫和一个时间T,迷宫中为X的格子是不能走的,现在给你起点和终点.问你能不能在正好T秒的时候从起点到达终点,**且不能走回头路.**
- 1 < N, M < 7; 0 < T < 50
- Sample Output
- NO
- YES

```
4 4 5
S.X.
..X.
..XD
....
3 4 5
S.X.
..X.
...D
0 0 0
```

HDU 1010 Tempter of the Bone

- 当前走到终点最少步数>满足条件还需要走的步数剪枝
- 奇偶剪枝
  可以把map看成这样：
  0 1 0
  1 0 1
  0 1 0
  0 ->1或1->0 必然是奇数步
  0->0 走1->1 必然是偶数步
  结论：
  所以当遇到从 0 走向 0 但是要求时间是奇数的，或者，
  从 1 走向 0 但是要求时间是偶数的 都可以直接判断不可达

优先队列

```
template<class T> using
minheap=
priority_queue<T, vector<T>,
greater<T>>;
template<class T> using
maxheap= priority_queue<T>;
```

优先队列

```
maxheap<int> m;
m.
```

```
top()                                  const int &
push(value_type &&__x)                       void
push(const value_type &__x)                  void
pop()                                        void
empty()                                      bool
size()                               unsigned long
emplace(_Args &&... __args)                  void
swap(priority_queue &__pq)                   void
```

POJ 2312 Battle City

- 题意：题目背景就是小时候玩的坦克大战，求从起点到终点最少需要多少步。已知S和R是不能走得，E是空的，可以走，B是砖，只有打掉后才可以通过。Y开始，T结束。每次操作可以是向四周射击或是向四周走一格。
- YBEB
- EERE
- SSTE

POJ 2312 Battle City

```
int dir[][2] = {{0,  1},{0,  -1},{1,  0},{-1, 0}};
char chess[N][N];
struct Node {
    int x, y, s;
    friend bool operator<(Node a, Node b)
        {return a.s > b.s; }
};
bool ok(int x, int y) {
    return  (x >= 0 && x < m && y >= 0 && y < n
            && chess[x][y] != 'S' && chess[x][y] != 'R');
}
```

POJ 2312 Battle City

```
int bfs() {
    priority_queue<Node> q; memset(visit, -1, sizeof(visit)); visit[sx][sy] =
0;q.push({sx, sy, 0}); while (!q.empty()) {
        Node f = q.top();q.pop();
        if (f.x == ex && f.y == ey)return f.s;
        for (int i = 0; i < 4; i++) {
            int dx = f.x + dir[i][0], dy = f.y + dir[i][1];
            if (ok(dx, dy) && visit[dx][dy]) {
                visit[dx][dy] = 0; q.push({dx, dy, f.s + 1 + (chess[dx][dy] ==
'B')});}}}return -1;}
```

没有了