程序设计实习

算法基础

张勤健
zqj@pku.edu.cn

北京大学信息科学技术学院

2024 年 5 月 22 日

搜索问题

北京飞往巴西里约热内卢

搜索问题

北京飞往巴西里约热内卢

好像没有直飞？怎么办？

搜索问题

- 状态空间
  - α: 未处理完的状态
  - β: 已处理的状态
- 初始状态
- 目标状态
- 后继函数
  有顺序的尝试备选动作，每一次的尝试都演化出另一个状态

影响搜索效率的因素

- 状态空间
  - α: 未处理完的状态
  - β: 已处理的状态
- 判重: 每次演化出一个状态 s 时, s 是否属于 α 或者 β
- 剪枝: 状态 s 的任意演化结果是否都属于 β
- 演化出来的状态数量: $\alpha \cup \beta$ 的大小

深度优先搜索 (Depth-First-Search)

从起点出发，走过的点要做标记，发现有没走过的点，就挑一个往前走，走不了就回退，此种路径搜索策略就称为“深度优先搜索”，简称“深搜”。

因为这种策略能往前走一步就往前走一步，总是试图走得更远。所谓远近 (或深度)，就是以距离起点的步数来衡量的。

在图上寻找路径

判断从 V 出发是否能走到终点：

```
bool dfs(V) {
  if (V 为终点) return true;
  if (V 为旧点) return false;
  将 V 标记为旧点;
  对和 V 相邻的每个节点 U {
    if (dfs(U) == true) return true;
  }
  return false;
}
int main() {
  将所有点都标记为新点;
  起点 = 1, 终点 = 8
  cout << dfs(起点) ;
  return 0;
}
```

在图上寻找路径

判断从 V 出发是否能走到终点，如果能，要记录路径：

```
Node path[MAX_LEN];  //MAX_LEN 取节点总数即可
int depth;
bool dfs(V) {
  if (V 为终点) {
    path[depth] = V;
    return true;
  }
  if (V 为旧点) return false;
  将 V 标记为旧点;
  path[depth]=V;
  ++depth;
  对和 V 相邻的每个节点 U {
    if (dfs(U) == true) return true;
  }
  --depth;
  return false;
}
int main() {
  将所有点都标记为新点;
  depth = 0;
  起点 = 1, 终点 = 8
  if (dfs(起点)) {
    for (int i = 0; i <= depth; ++ i)
      cout << path[i] << endl;
  }
  return 0;
}
```

在图上寻找最优 (步数最少) 路径

```
Node bestPath[MAX_LEN];
int minSteps = INFINITE; //最优路径步数
Node path[MAX_LEN];  //MAX_LEN 取节点总数即可
int depth;
void dfs(V) {
  if (V 为终点) {
    path[depth] = V;
    if (depth < minSteps) {
      minSteps = depth;
      拷贝 path 到 bestPath;
    }
    return;
  }
  if (V 为旧点)  return;
  if (depth >= minSteps) return ; //最优性剪枝
  将 V 标记为旧点;
  path[depth]=V;
  ++depth;
  对和 V 相邻的每个节点 U {
    dfs(U);
  }
  将 V 恢复为新点
  --depth;
}
int main() {
  将所有点都标记为新点;
  depth = 0;
  dfs(起点);
  if (minSteps != INFINITE) {
    for (int i = 0; i <= minSteps; ++ i)
      cout << bestPath[i] << endl;
  }
}
```

遍历图上所有节点

```
dfs(V) {
  if( V 是旧点) return;
  将 V 标记为旧点;
  对和 V 相邻的每个点 U {
    dfs(U);
  }
}
int main() {
  将所有点都标记为新点;
  while(在图中能找到新点 k)
    dfs(k);
}
```

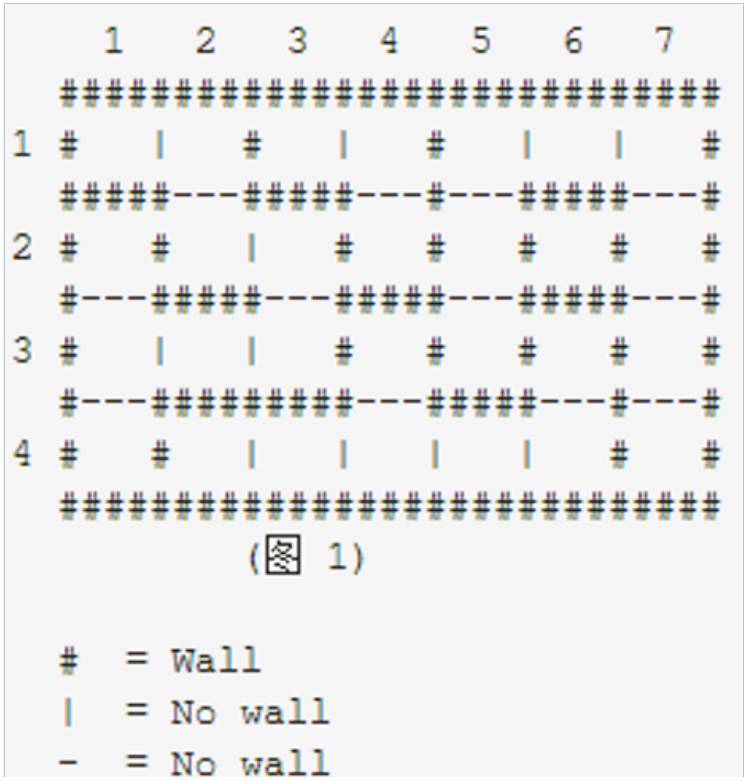
图的表示方法

邻接矩阵
用一个二维数组 G 存放图，$G[i][j]$ 表示节点 i 和节点 j 之间边的情况 (如有无边，边方向，权值大小等) 。
遍历复杂度：$O(n^2)$,n 为节点数目

邻接表
每个节点 V 对应一个一维数组 (vector)，里面存放从 V 连出去的边，边的信息包括另一顶点，还可能包含边权值等。
遍历复杂度：$O(n+e)$,n 为节点数目，e 为边数目

例题 01 城堡问题

右图是一个城堡的地形图。请你编写一个程序，计算城堡一共有多少房间，最大的房间有多大。城堡被分割成 $m \times n (m \leq 50, n \leq 50)$ 个方块，每个方块可以有 $0 \sim 4$ 面墙。



例题 01 城堡问题

- 输入
  第一行是两个整数，分别是南北向、东西向的方块数。
  在接下来的输入行里，每个方块用一个数字 $(0 \leq p \leq 50)$ 描述。用一个数字表示方块周围的墙，1 表示西墙，2 表示北墙，4 表示东墙，8 表示南墙。每个方块用代表其周围墙的数字之和表示。城堡的内墙被计算两次，方块 $(1,1)$ 的南墙同时也是方块 $(2,1)$ 的北墙。
  输入的数据保证城堡至少有两个房间。
- 输出
  城堡的房间数、城堡中最大房间所包括的方块数。
- 样例输入

```
4
7
11 6 11 6 3 10 6
7 9 6 13 5 15 5
1 10 12 7 13 7 5
13 11 10 8 10 12 13
```

- 样例输出

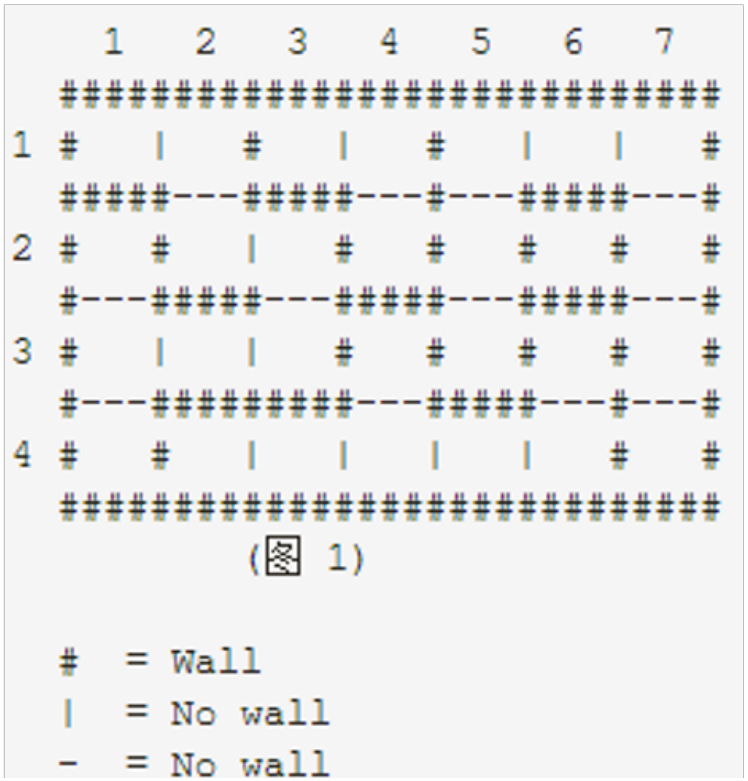
```
5
9
```

例题 01 城堡问题

对每一个房间，扩展相邻的房间，从而给这个房间能够到达的所有位置染色。最后统计一共用了几种颜色，以及每种颜色的数量。
比如:

1 1 2 2 3 3 3
1 1 1 2 3 4 3
1 1 1 5 3 5 3
1 5 5 5 5 5 3

从而一共有 5 个房间，最大的房间 (1) 占据 9 个格子



例题 01 城堡问题

```cpp
#include <iostream>
#include <cstring>
using namespace std;
int R, C, maxRoomArea = 0, roomNum = 0, roomArea;
int rooms[60][60], color[60][60]; //color 表示方块是否染色过的标记
void dfs(int i, int j) {
  if (color[i][j]) return;
  ++ roomArea;
  color[i][j] = roomNum;
  if ((rooms[i][j] & 1) == 0) dfs(i, j-1);   //向西走
  if ((rooms[i][j] & 2) == 0) dfs(i-1, j);   //向北
  if ((rooms[i][j] & 4) == 0) dfs(i, j+1);  //向东
  if ((rooms[i][j] & 8) == 0) dfs(i+1, j);  //向南
}
int main() {
  cin >> R >> C;
  for (int i = 1; i <= R; ++i)
    for (int j = 1; j <= C; ++j)
      cin >> rooms[i][j];
  memset(color,0,sizeof(color));
  for (int i = 1; i <= R; ++i) {
    for (int j = 1; j <= C; ++j) {
      if (!color[i][j]) {
        ++ roomNum;
        roomArea = 0;
        dfs(i, j);
        maxRoomArea = max(roomArea, maxRoomArea);
      }
    }
  }
  cout << roomNum << endl << maxRoomArea << endl;
  return 0;
}
```

例题 01 城堡问题-非递归

```
1  #include <iostream>
2  #include <stack>
3  #include <cstring>
4  using namespace std;
5  int R, C, maxRoomArea = 0, roomNum = 0, roomArea;
6  int rooms[60][60], color[60][60]; // color 表示方块是否染色过的标记
7  void dfs(int r, int c) {
8    struct Room {
9      int r, c;
10     Room(int rr, int cc) : r(rr), c(cc) {}
11   };
12   stack<Room> stk;
13   stk.push(Room(r, c));
14   while (!stk.empty()) {
15     Room rm = stk.top();
16     int i = rm.r, j = rm.c;
17     if (color[i][j]) {
18       stk.pop();
19       continue;
20     }
21     ++roomArea;
22     color[i][j] = roomNum;
23     if ((rooms[i][j] & 1) == 0) stk.push(Room(i, j - 1)); //向西
24     if ((rooms[i][j] & 2) == 0) stk.push(Room(i - 1, j)); //向北
25     if ((rooms[i][j] & 4) == 0) stk.push(Room(i, j + 1)); //向东
26     if ((rooms[i][j] & 8) == 0) stk.push(Room(i + 1, j)); //向南
27   }
28 }
```

例题 02 踩方格

有一个方格矩阵，矩阵边界在无穷远处。我们做如下假设:

1. 每走一步时，只能从当前方格移动一格，走到某个相邻的方格上；
2. 走过的格子立即塌陷无法再走第二次
3. 只能向北、东、西三个方向走

请问：如果允许在方格矩阵上走 n 步 $(n \leq 20)$，共有多少种不同的方案。2 种走法只要有一步不一样，即被认为是不同的方案。

例题 02 踩方格

思路：
递归
从 (i, j) 出发，走 n 步的方案数，等于以下三项之和：

从 $(i+1, j)$ 出发，走 $n-1$ 步的方案数。前提：$(i+1, j)$ 还没走过
从 $(i, j+1)$ 出发，走 $n-1$ 步的方案数。前提：$(i, j+1)$ 还没走过
从 $(i, j-1)$ 出发，走 $n-1$ 步的方案数。前提：$(i, j-1)$ 还没走过

例题 02 踩方格

```cpp
#include <iostream>
#include <cstring>
using namespace std;
int visited[30][50];
int ways( int i,int j,int n) {
  if (n == 0) return 1;
  visited[i][j] = 1;
  int num = 0;
  if (!visited[i][j-1]) num+= ways(i,j-1,n-1);
  if (!visited[i][j+1]) num+= ways(i,j+1,n-1);
  if (!visited[i+1][j]) num+= ways(i+1,j,n-1);
  visited[i][j] = 0;
  return num;
}
int main() {
  int n;
  cin >> n;
  memset(visited,0,sizeof(visited));
  cout << ways(0,25,n) << endl;
  return 0;
}
```

例题 02 踩方格

$l(n)$ 表示最后一步向西走走 n 步的方案数，
$r(n)$ 表示最后一步向东走走 n 步的方案数，
$u(n)$ 表示最后一步走北走走 n 步的方案数；
则

$$l(n) = l(n-1) + u(n-1)$$

$$r(n) = r(n-1) + u(n-1)$$

$$u(n) = l(n-1) + r(n-1) + u(n-1)$$

$f(n)$ 表示走 n 步的方案数，则

$$\begin{aligned}
f(n) &= l(n) + r(n) + u(n) \\
&= l(n-1) + u(n-1) + r(n-1) + u(n-1) + l(n-1) + r(n-1) + u(n-1) \\
&= 2l(n-1) + 2r(n-1) + 3u(n-1) \\
&= 2 * (l(n-1) + r(n-1) + u(n-1)) + u(n-1) \\
&= 2f(n-1) + l(n-2) + r(n-2) + u(n-2) \\
&= 2f(n-1) + f(n-2)
\end{aligned}$$

例题 03 Roads

N 个城市，编号 1 到 N。城市间有 R 条单向道路。
每条道路连接两个城市，有长度和过路费两个属性。
Bob 只有 K 块钱，他想从城市 1 走到城市 N。问最短共需要走多长的路。如果到不了 N，输出 -1

$2 \leq N \leq 100$
$0 \leq K \leq 10000$
$1 \leq R \leq 10000$
每条路的长度 L, $1 \leq L \leq 100$
每条路的过路费 T , $0 \leq T \leq 100$

例题 03 Roads

```
1  #include <iostream>
2  #include <vector>
3  #include <cstring>
4  using namespace std;
5  int K,N,R;
6  struct Road {
7    int d,L,t;
8  };
9  vector<vector<Road> > cityMap(110);//邻接表。cityMap[i] 是从点 i 有路连到的城市集合
10 int minLen = 1 << 30;  //当前找到的最优路径的长度
11 int totalLen;   //正在走的路径的长度
12 int totalCost ; //正在走的路径的花销
13 int visited[110];  //城市是否已经走过的标记
```

例题 03 Roads

```
14
15  void dfs(int s) { //从 s 开始向 N 行走
16    if (s == N) {
17      minLen = min(minLen, totalLen);
18      return ;
19    }
20    for (int i = 0 ;i < cityMap[s].size(); ++i) {
21      int d = cityMap[s][i].d;  //s 有路连到 d
22      if (visited[d]) continue;
23      int cost = totalCost + cityMap[s][i].t;
24      if (cost > K) continue;
25      totalLen += cityMap[s][i].L;
26      totalCost +=  cityMap[s][i].t;
27      visited[d] = 1;
28      dfs(d);
29      visited[d] = 0;
30      totalCost -=  cityMap[s][i].t;
31      totalLen -= cityMap[s][i].L;
32    }
33  }
```

例题 03 Roads

```
34
35  int main() {
36    cin >>K >> N >> R;
37    for (int i = 0;i < R; ++i) {
38      int s;
39      Road r;
40      cin >> s >> r.d >> r.L >> r.t;
41      if( s != r.d ) cityMap[s].push_back(r);
42    }
43    memset(visited,0,sizeof(visited));
44    totalLen = 0;
45    totalCost = 0;
46    visited[1] = 1;
47    minLen = 1 << 30;
48    dfs(1);
49    if (minLen < (1 << 30))
50      cout << minLen << endl;
51    else
52      cout << "-1" << endl;
53    return 0;
54  }
```

例题 03 Roads-剪枝

最优性剪枝：

1. 如果当前已经找到的最优路径长度为 L，那么在继续搜索的过程中，总长度已经大于等于 L 的走法，就可以直接放弃，不用走到底了
2. 保存中间计算结果用于最优性剪枝：
   用 $minL[k][m]$ 表示：走到城市 k 时总过路费为 m 的条件下，最优路径的长度。若在后续的搜索中，再次走到 k 时，如果总路费恰好为 m，且此时的路径长度已经不小于 $minL[k][m]$，则不必再走下去了。

例题 03 Roads-剪枝

```
15  int minL[110][10100]; //minL[i][j] 表示从 1 到 i 点的，花销为 j 的最短路的长度
16  void dfs(int s) { //从 s 开始向 N 行走
17    if (s == N) {
18      minLen = min(minLen, totalLen);
19      return ;
20    }
21    for (int i = 0 ;i < cityMap[s].size(); ++i) {
22      int d = cityMap[s][i].d;  //s 有路连到 d
23      if (visited[d]) continue;
24      int cost = totalCost +  cityMap[s][i].t;
25      if (cost > K) continue;
26      if (totalLen + cityMap[s][i].L >= minLen) continue;//剪枝 1
27      if (totalLen + cityMap[s][i].L >= minL[d][cost]) continue;//剪枝 2
28      totalLen += cityMap[s][i].L;
29      totalCost += cityMap[s][i].t;
30      minL[d][cost] = totalLen;
31      visited[d] = 1;
32      dfs(d);
33      visited[d] = 0;
34      totalCost -=  cityMap[s][i].t;
35      totalLen -= cityMap[s][i].L;
36    }
37  }
```

例题 03 Roads-剪枝

```
39  int main() {
40    cin >>K >> N >> R;
41    for (int i = 0;i < R; ++i) {
42      int s;
43      Road r;
44      cin >> s >> r.d >> r.L >> r.t;
45      if( s != r.d ) cityMap[s].push_back(r);
46    }
47    for (int i = 0;i < 110; ++i)
48      for (int j = 0; j < 10100; ++j)
49        minL[i][j] = 1 << 30;
50    memset(visited,0,sizeof(visited));
51    totalLen = 0;
52    totalCost = 0;
53    visited[1] = 1;
54    minLen = 1 << 30;
55    dfs(1);
56    if (minLen < (1 << 30))
57      cout << minLen << endl;
58    else
59      cout << "-1" << endl;
60    return 0;
61  }
```

例题 03 Roads-剪枝

如果到达某个状态 A 时，发现前面曾经也到达过 A，且前面那次到达 A 所花代价更少，则剪枝。这要求保存到达状态 A 的到目前为止的最少代价。

用 $minL[k][m]$ 表示：走到城市 k 时总过路费为 m 的条件下，最优路径的长度。若在后续的搜索中，再次走到 k 时，如果总路费恰好为 m，且此时的路径长度已经不小于 $minL[k][m]$, 则不必再走下去了。

例题 04 生日蛋糕

要制作一个体积为 $N\pi$ 的 M 层生日蛋糕，每层都是一个圆柱体。
设从下往上数第 $i(1 \le i \le M)$ 层蛋糕是半径为 R_i, 高度为 H_i 的圆柱。
当 $i < M$ 时，要求 $R_i > R_{i+1}$ 且 $H_i > H_{i+1}$。
由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积 Q 最小。
令 $Q = S\pi$
请编程对给出的 N 和 M，找出蛋糕的制作方案（适当的 R_i 和 H_i 的值），使 S 最小。（除 Q 外，以上所有数据皆为正整数）

例题 04 生日蛋糕

解题思路

- 深度优先搜索，枚举什么？

例题 04 生日蛋糕

解题思路

- 深度优先搜索，枚举什么？
  枚举每一层可能的高度和半径。

例题 04 生日蛋糕

解题思路

- 深度优先搜索，枚举什么？
  枚举每一层可能的高度和半径。
- 如何确定搜索范围？

例题 04 生日蛋糕

解题思路

- 深度优先搜索，枚举什么?
  枚举每一层可能的高度和半径。
- 如何确定搜索范围?
  底层蛋糕的最大可能半径和最大可能高度

例题 04 生日蛋糕

解题思路

- 深度优先搜索，枚举什么?
  枚举每一层可能的高度和半径。
- 如何确定搜索范围?
  底层蛋糕的最大可能半径和最大可能高度
- 搜索顺序，哪些地方体现搜索顺序?

例题 04 生日蛋糕

解题思路

- 深度优先搜索，枚举什么？
  枚举每一层可能的高度和半径。
- 如何确定搜索范围？
  底层蛋糕的最大可能半径和最大可能高度
- 搜索顺序，哪些地方体现搜索顺序？
  - 从底层往上搭蛋糕，而不是从顶层往下搭
  - 在同一层进行尝试的时候，半径和高度都是从大到小试

例题 04 生日蛋糕

```cpp
#include <iostream>
#include <cmath>
using namespace std;
int N,M;
int minArea = 1 << 30; //最优表面积
int area = 0; //正在搭建中的蛋糕的表面积
int minV[30] = {0}; // minV[n] 表示 n 层蛋糕最少的体积
void dfs(int v, int n, int r, int h) {
  //要用 n 层去凑体积 v, 最底层半径不能超过 r, 高度不能超过 h
  //求出最小表面积放入 minArea
  if (n == 0) {
    if (v == 0) minArea = min(minArea, area);
    return;
  }
  if (v <= 0) return ;
  for (int rr = r; rr >=n; --rr) {
    if (n == M) area = rr * rr;//底面积
    for (int hh = h; hh >= n ; --hh) {
      area += 2 * rr * hh;
      dfs(v - rr * rr * hh, n - 1, rr - 1, hh - 1);
      area -= 2 * rr * hh;
    }
  }
}
```

例题 04 生日蛋糕

```
25
26  int main() {
27    cin >> N >> M  ;//M 层蛋糕，体积 N
28    for (int i = 1; i <= M; ++i) {
29      minV[i] = minV[i - 1] + i * i * i; //第 i 层半径至少 i，高度至少 i
30    }
31    if (minV[M] > N) {
32      cout << 0 << endl;
33      return 0;
34    }
35    int maxH =  (N - minV[M - 1]) / (M * M) + 1;//底层最大高度
36    //最底层体积不超过 (N-minV[M-1])，且半径至少 M
37    int maxR = sqrt(double(N - minV[M - 1]) / M) + 1;//底层高度至少 M
38    dfs(N, M, maxR, maxH);
39    if (minArea == 1 << 30)
40      cout << 0 << endl;
41    else
42      cout << minArea << endl;
43    return 0;
44  }
```

例题 04 生日蛋糕-剪枝

1. 剪枝 1：搭建过程中发现已建好的面积已经不小于目前求得的最优表面积，或者预见到搭完后面积一定会不小于目前最优表面积，则停止搭建（最优性剪枝）

例题 04 生日蛋糕-剪枝

1. 剪枝 1：搭建过程中发现已建好的面积已经不小于目前求得的最优表面积，或者预见到搭完后面积一定会不小于目前最优表面积，则停止搭建（最优性剪枝）
2. 剪枝 2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建 (可行性剪枝)

例题 04 生日蛋糕-剪枝

1. 剪枝 1：搭建过程中发现已建好的面积已经不小于目前求得的最优表面积，或者预见到搭完后面积一定会不小于目前最优表面积，则停止搭建（最优性剪枝）
2. 剪枝 2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建 (可行性剪枝)
3. 剪枝 3：搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建 (可行性剪枝)

例题 04 生日蛋糕-剪枝

1. 剪枝 1：搭建过程中发现已建好的面积已经不小于目前求得的最优表面积，或者预见到搭完后面积一定会不小于目前最优表面积，则停止搭建（最优性剪枝）
2. 剪枝 2：搭建过程中预见到再往上搭，高度已经无法安排，或者半径已经无法安排，则停止搭建 (可行性剪枝)
3. 剪枝 3：搭建过程中发现还没搭的那些层的体积，一定会超过还缺的体积，则停止搭建 (可行性剪枝)
4. 剪枝 4：搭建过程中发现还没搭的那些层的体积，最大也到不了还缺的体积，则停止搭建 (可行性剪枝)

例题 04 生日蛋糕-剪枝

```cpp
#include <iostream>
#include <vector>
#include <cstring>
#include <cmath>
using namespace std;
int N,M;
int minArea = 1 << 30; //最优表面积
int area = 0; //正在搭建中的蛋糕的表面积
int minV[30] = {0}; // minV[n] 表示 n 层蛋糕最少的体积
int minA[30] = {0}; // minA[n] 表示 n 层蛋糕的最少侧表面积
int MaxVforNRH(int n, int r, int h) {
  //求在 n 层蛋糕，底层最大半径 r，最高高度 h 的情况下，能凑出来的最大体积
  int v = 0;
  for (int i = 0; i < n; ++i)
    v += (r - i) * (r - i) * (h - i);
  return v;
}
```

例题 04 生日蛋糕-剪枝

```
18
19  void dfs(int v, int n,int r,int h) {
20    //要用 n 层去凑体积 v, 最底层半径不能超过 r, 高度不能超过 h , 求出最小表面积放入 m
21    if (n == 0) {
22      if(v == 0) minArea = min(minArea, area);
23      return;
24    }
25    if (v <= 0) return ;
26    if (minV[n] > v) return ;//剪枝 3
27    if (area + minA[n] >= minArea) return ;//剪枝 1
28    if (h < n || r < n) return ;//剪枝 2
29    if (MaxVforNRH(n, r, h) < v)  return;//剪枝 4
30    //这个剪枝最强! 没有的话, 5 秒都超时, 有的话, 10ms 过!
31
32    //for( int rr = n; rr <= r; ++ rr ) { 这种写法比从大到小慢 5 倍
33    for (int rr = r; rr >= n; --rr) {
34      if (n == M) area = rr * rr;//底面积
35      for (int hh = h; hh >= n; --hh) {
36        area += 2 * rr * hh;
37        dfs(v-rr*rr*hh, n-1, rr-1, hh-1);
38        area -= 2 * rr * hh;
39      }
40    }
41  }
```

例题 04 生日蛋糕-剪枝

```
42
43  int main() {
44    cin >> N >> M  ;//M 层蛋糕，体积 N
45    for (int i = 1; i<= M; ++ i) {
46      minV[i] = minV[i-1] + i * i * i; //第 i 层半径至少 i，高度至少 i
47      minA[i] =  minA[i-1] + 2 * i * i;
48    }
49    if (minV[M] > N) {
50      cout << 0 << endl;
51      return 0;
52    }
53    int maxH =  (N - minV[M - 1]) / (M * M) + 1;//底层最大高度
54    //最底层体积不超过 (N-minV[M-1])，且半径至少 M
55    int maxR = sqrt(double(N - minV[M - 1]) / M) + 1;//底层高度至少 M
56    dfs(N, M, maxR, maxH);
57    if (minArea == 1 << 30)
58      cout << 0 << endl;
59    else
60      cout << minArea << endl;
61    return 0;
62  }
```

例题 04 生日蛋糕

还有什么可以改进

- 用数组存放 $MaxVforNRH(n, r, h)$ 的计算结果，避免重复计算

例题 04 生日蛋糕

还有什么可以改进

- 用数组存放 $MaxVforNRH(n, r, h)$ 的计算结果，避免重复计算
- 加上对 dfs 失败原因的判断。

```
for (int rr = r; rr >= n; --rr) {
  if (n == M) area = rr * rr;//底面积
  for (int hh = h; hh >= n; --hh) {
    area += 2 * rr * hh;
    dfs(v-rr*rr*hh, n-1, rr-1, hh-1);
    //如果是因为剩余体积不够大而失败，那么就用不着试下一个高度，直接 break;
    area -= 2 * rr * hh;
  }
}
```

例题 05 木棒

乔治拿来一组等长的木棒，将它们随机地裁断，使得每一节木棍的长度都不超过 50 个长度单位。然后他又想把这些木棍恢复到为裁截前的状态，但忘记了初始时有多少木棒以及木棒的初始长度。请你设计一个程序，帮助乔治计算木棒的可能最小长度。每一节木棍的长度都用大于零的整数表示。

输入
输入包含多组数据，每组数据包括两行。第一行是一个不超过 64 的整数，表示砍断之后共有多少节木棍。第二行是截断以后，所得到的各节木棍的长度。在最后一组数据之后，是一个零。
输出
为每组数据，分别输出原始木棒的可能最小长度，每组数据占一行。

样例输入：

```
9
5 2 1 5 2 1 5 2 1
4
1 2 3 4
0
```

样例输出：

```
6
5
```

例题 05 木棒

尝试 (枚举) 什么?

例题 05 木棒

尝试 (枚举) 什么?

- 枚举所有可能的棍子长度

例题 05 木棒

尝试 (枚举) 什么?

- 枚举所有可能的棍子长度
- 从最长的那根木棒的长度一直枚举到木棒长度总和的一半

例题 05 木棒

尝试 (枚举) 什么?

- 枚举所有可能的棍子长度
- 从最长的那根木棒的长度一直枚举到木棒长度总和的一半
- 对每个假设的棍子长度，试试看能否拼齐若干根棍子

例题 05 木棒

尝试 (枚举) 什么?

- 枚举所有可能的棍子长度
- 从最长的那根木棒的长度一直枚举到木棒长度总和的一半
- 对每个假设的棍子长度，试试看能否拼齐若干根棍子

真的要每个长度都试吗?

例题 05 木棒

尝试 (枚举) 什么?

- 枚举所有可能的棍子长度
- 从最长的那根木棒的长度一直枚举到木棒长度总和的一半
- 对每个假设的棍子长度，试试看能否拼齐若干根棍子

真的要每个长度都试吗?

- 对于不是木棒总长度的因子的长度，可以直接否定，不需尝试

例题 05 木棒

假设了一个棍子长度的前提下，如何尝试去拼成若干根该长度的棍子?

例题 05 木棒

假设了一个棍子长度的前提下，如何尝试去拼成若干根该长度的棍子?

- 一根一根地拼棍子

例题 05 木棒

假设了一个棍子长度的前提下，如何尝试去拼成若干根该长度的棍子?

- 一根一根地拼棍子
- 如果拼好前 i 根棍子，结果发现第 $i+1$ 根无论如何拼不成了

例题 05 木棒

假设了一个棍子长度的前提下，如何尝试去拼成若干根该长度的棍子?

- 一根一根地拼棍子
- 如果拼好前 i 根棍子，结果发现第 $i+1$ 根无论如何拼不成了 -> 推翻第 i 根的拼法，重拼第 i 根
- 直至有可能推翻第 1 根棍子的拼法

例题 05 木棒

假设了一个棍子长度的前提下，如何尝试去拼成若干根该长度的棍子?

- 一根一根地拼棍子
- 如果拼好前 i 根棍子，结果发现第 $i+1$ 根无论如何拼不成了 -> 推翻第 i 根的拼法，重拼第 i 根
- 直至有可能推翻第 1 根棍子的拼法

例题 05 木棒

本题的“状态”是什么?

例题 05 木棒

本题的“状态”是什么?

- 状态可以是一个二元组 (R, M)

例题 05 木棒

本题的“状态”是什么?

- 状态可以是一个二元组 (R, M)
- R：还没被用掉的木棒数目
- M：当前正在拼的棍子还缺少的长度

例题 05 木棒

本题的“状态”是什么?

- 状态可以是一个二元组 (R, M)
- R：还没被用掉的木棒数目
- M：当前正在拼的棍子还缺少的长度

初始状态和搜索的目标状态 (解状态) 是什么?

例题 05 木棒

本题的“状态”是什么?

- 状态可以是一个二元组 (R, M)
- R：还没被用掉的木棒数目
- M：当前正在拼的棍子还缺少的长度

初始状态和搜索的目标状态 (解状态）是什么?
假设共有 N 节木棒，假定的棍子长度是 L:

- 初始状态：(N, L)
- 目标状态：$(0, 0)$

例题 05 木棒

本题的“状态”是什么?

- 状态可以是一个二元组 (R, M)
- R：还没被用掉的木棒数目
- M：当前正在拼的棍子还缺少的长度

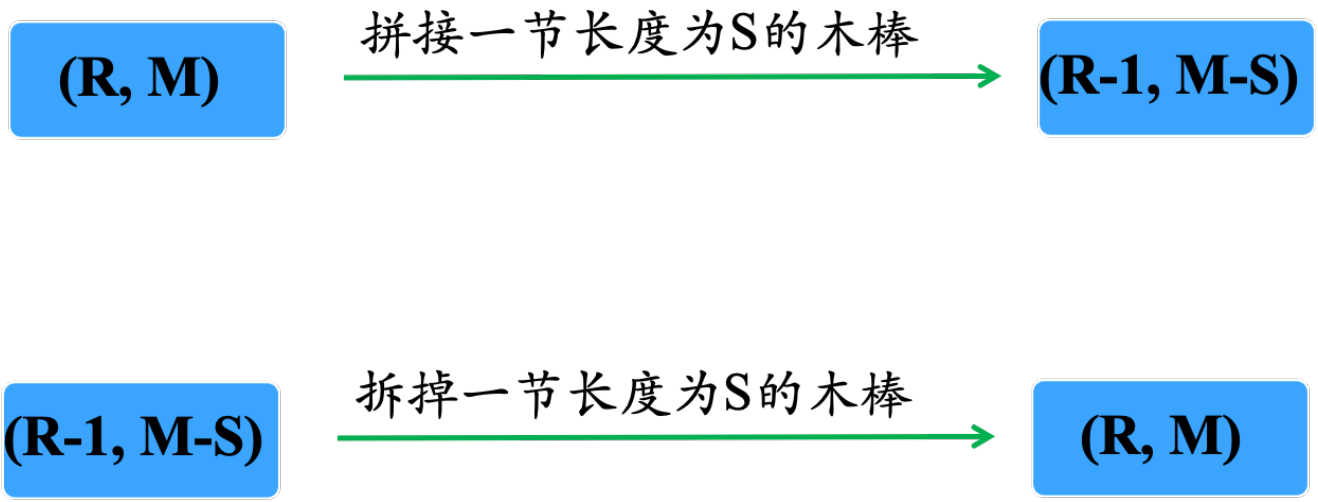
初始状态和搜索的目标状态 (解状态）是什么?
假设共有 N 节木棒，假定的棍子长度是 L:

- 初始状态：(N, L)
- 目标状态：$(0, 0)$

所谓“成功拼出若干根长度为 L 的棍子”，就是要在状态空间中找到一条从 (N, L) 到 $(0, 0)$ 的路径

例题 05 木棒

“状态”之间如何转移?



例题 05 木棒

```cpp
#include <iostream>
#include <memory.h>
#include <cstdlib>
#include <vector>
#include <algorithm>
using namespace std;
int N, L;
vector<int> length;
int used[65];  //是否用过的标记
int i, j, k;
int dfs(int R, int M)  {
// M 表示当前正在拼的棍子和 L 比还缺的长度
  if (R == 0 && M == 0) return true;
  if (M == 0) M = L; //一根刚刚拼完，开始拼新的一根
  for (int i = 0; i < N; i++) {
    if (!used[i] && length[i] <= M) {
      used[i] = 1;
      if (dfs(R - 1, M - length[i])) return true;
      used[i] = 0; //说明本次不能用第 i 根，第 i 根以后还有用
    }
  }
  return false;
}
```

例题 05 木棒

```
24
25  int main() {
26    while(cin >> N && N != 0) {
27      int totalLen = 0;
28      length.clear();
29      for(int i = 0; i < N; i ++ ) {
30        int n;
31        cin >> n;
32        length.push_back(n);
33        totalLen += length[i];
34      }
35      sort(length.begin(), length.end(), greater<int>()); //要从长到短进行尝试
36      for (L = length[0]; L <= totalLen / 2; L++) {
37        if (totalLen % L) continue;
38        memset(used, 0,sizeof(used));
39        if (dfs(N,L)) {
40          cout << L << endl;
41          break;
42        }
43      }
44      if (L > totalLen / 2) cout << totalLen << endl;
45    } // while
46    return 0;
47  }
```

例题 05 木棒

慢！怎么办?

例题 05 木棒

慢！怎么办?
剪枝

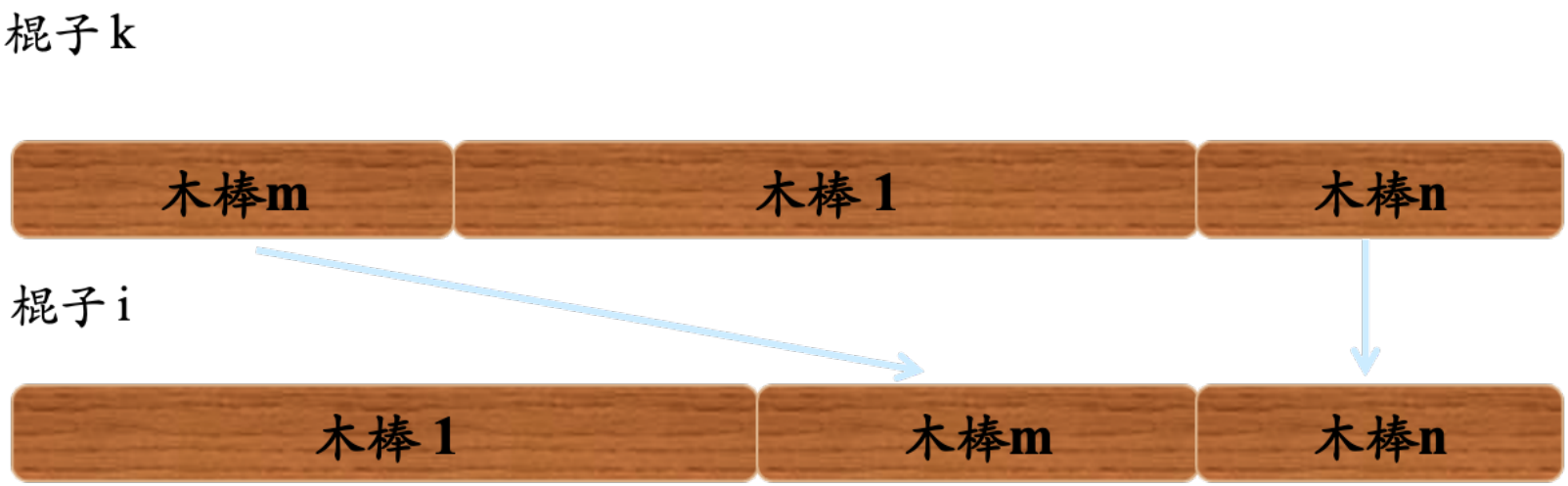
例题 05 木棒-剪枝

第一种剪枝方案:
不要在同一个位置多次尝试相同长度的木棒

即：如果某次拼接选择长度为 S 的木棒，导致最终失败，则在同一位置尝试下一根木棒时，要跳过所有长度为 S 的木棒

例题 05 木棒-剪枝

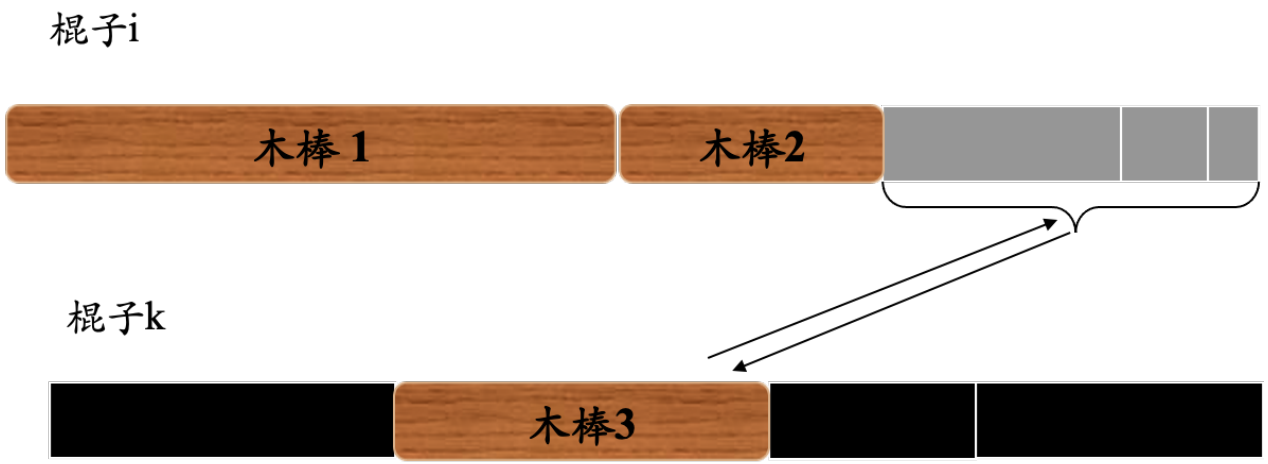
第二种剪枝方案:
如果由于以后的拼接失败，需要重新调整第 i 根棍子的拼法，则不会考虑替换第 i 根棍子中的第一根木棒（换了也没用）
为什么?
因为假设替换后能全部拼成功，那么这被换下来的第一根木棒，必然会出现在以后拼好的某根棍子 k 中
那么我们原先拼第 i 根棍子时, 就可以用和棍子 k 同样的构成法来拼，照这种构成法拼好第 i 根棍子，继续下去最终也应该能够全部拼成功



例题 05 木棒-剪枝

第三种剪枝方案:
不要希望通过仅仅替换已拼好棍子的最后一根木棒就能够改变失败的局面

为什么?
假设替换 3 后最终能够成功，那么 3 必然出现在后面的某个棍子 k 里
将棍子 k 中的 3 和棍子 i 中用来替换 3 的几根木棒对调，结果当然一样是成功的
这就和 i 原来的拼法会导致不成功矛盾



例题 05 木棒-剪枝

第四种剪枝方案:
拼每一根棍子的时候，应该确保已经拼好的部分，长度是从长到短排列的，即拼的过程中要排除类似下面这种情况:
未完成的棍子 i

木棒 3 比木棒 2 长，这种情况的出现是一种浪费。因为要是这样往下能成功，那么 $2,3$ 对调的拼法肯定也能成功。由于取木棒是从长到短的，所以能走到这一步，就意味着当初将 3 放在 2 的位置时，是不成功的

例题 05 木棒-剪枝

第四种剪枝方案:
拼每一根棍子的时候，应该确保已经拼好的部分，长度是从长到短排列的，即拼的过程中要排除类似下面这种情况：
未完成的棍子 i

木棒 3 比木棒 2 长，这种情况的出现是一种浪费。因为要是这样往下能成功，那么 $2,3$ 对调的拼法肯定也能成功。由于取木棒是从长到短的，所以能走到这一步，就意味着当初将 3 放在 2 的位置时，是不成功的

排除办法：每次找一根木棒的时候，只要这不是一根棍子的第一条木棒，就不应该从下标为 0 的木棒开始找，而应该从刚刚 (最近）接上去的那条木棒的下一条开始找。这样，就不会往 2 后面接更长的 3 了
为此，要设置一个全局变量 $lastStickNo$(初值 -1)，记住最近拼上去的那条木棒的下标。

例题 05 木棒-剪枝

```
11  int lastStickNo = -1;
12  int dfs( int R, int M) {
13  // M 表示当前正在拼的棍子和 L 比还缺的长度
14    if( R == 0 && M == 0 ) return true;
15    if( M == 0 ) M = L; //一根刚刚拼完，开始拼新的一根
16    int startNo = 0;
17    if( M != L ) startNo = lastStickNo + 1;//剪枝 4
18    for( int i = startNo; i < N; i ++) {
19      if( !used[i] && length[i] <= M) {
20        //剪枝 1
21        if(i > 0 && used[i-1] == false && length[i] == length[i-1]) continue;
22        used[i] = 1;
23        lastStickNo = i;
24        if (dfs(R- 1, M - length[i])) return true;
25        used[i] = 0; //说明本次不能用第 i 根，第 i 根以后还有用
26        if (length[i] == M || M == L) return false; //剪枝 3, 2
27      }
28    }
29    return false;
30  }
```

例题 05 木棒-剪枝

1. 要选择合适的搜索顺序
   如果一个任务分为 $A, B, C, \ldots$ 等步骤 (先后次序无关)，要优先尝试可能性少的步骤
2. 要发现表面上不同，实质相同的重复状态，避免重复的搜索
3. 要根据实际问题发掘剪枝方案