Treinos Livres
Maratona de Programação

André Augusto
Humberto Longo
Paulo Cezar Pereira Costa

Instituto de Informática
Universidade Federal de Goiás

5 de abril de 2013

Aula 4
Complete Search - Busca exaustiva

Complete Search

- Estratégia baseada no princípio KISS ("Keep It Simple, Stupid")
  - Buscar os resultados evitando qualquer complexidade desnecessária.
- O objetivo numa competição de programação é escrever um programa que resolva o problema dentro do tempo limite.
  - Não importa se existe ou não uma solução mais eficiente.
- A busca exaustiva faz uso do método trivial, de força bruta, todas as possíveis soluções são analisadas para encontrar a resposta.
- Essa técnica sempre deve ser a primeira a ser considerada.
  - Caso funcione dentro do limite de tempo / memória, use-a!
    - Geralmente é fácil de codificar e debugar.
- Apenas alguns milhões de possíveis respostas para um problema? Itere em todas elas e encontre aquela que funciona.

Complete Search

- Estratégia baseada no princípio KISS ("Keep It Simple, Stupid")
  - Buscar os resultados evitando qualquer complexidade desnecessária.
- O objetivo numa competição de programação é escrever um programa que resolva o problema dentro do tempo limite.
  - Não importa se existe ou não uma solução mais eficiente.
- A busca exaustiva faz uso do método trivial, de força bruta, todas as possíveis soluções são analisadas para encontrar a resposta.
- Essa técnica sempre deve ser a primeira a ser considerada.
  - Caso funcione dentro do limite de tempo / memória, use-a!
    - Geralmente é fácil de codificar e debugar.
- Apenas alguns milhões de possíveis respostas para um problema? Itere em todas elas e encontre aquela que funciona.

Cuidado!!

Nem sempre é óbvio que a busca exaustiva pode ser usada.

Complete Search

- Uma das técnicas de resolução de problemas mais importantes;
  - Pode ser aplicada a uma grande gama de problemas quando as instâncias são pequenas os suficiente
  - Ponto de partida para o desenvolvimento de outros algoritmos.
- Competidor precisa saber:
  - Gerar/testar: subconjuntos, permutações, ...
  - Técnicas para reduzir o espaço de busca
  - Estimar a complexidade no pior caso
- Ajustes no código podem influenciar bastante no tempo de execução;
  - Vale a pena implementar a mesma solução de formas diferentes.

#protip

Quando não conseguir pensar em um jeito melhor de resolver o problema arrisque a solução por força bruta. Caso exista um caso onde ela não é rápida o suficiente, mantenha a solução por perto e use-a para testar outras soluções nos casos menores.

Filtrar vs. Gerar

Duas possíveis abordagens podem ser escolhidas quando fazendo uma busca exaustiva:

- Filtragem – Todas as possíveis soluções são geradas e depois examinadas para eliminar as inválidas.
- Geração – Soluções construídas progressivamente, assim que uma inconsistência é detectada a solução é descartada.

Complete Search

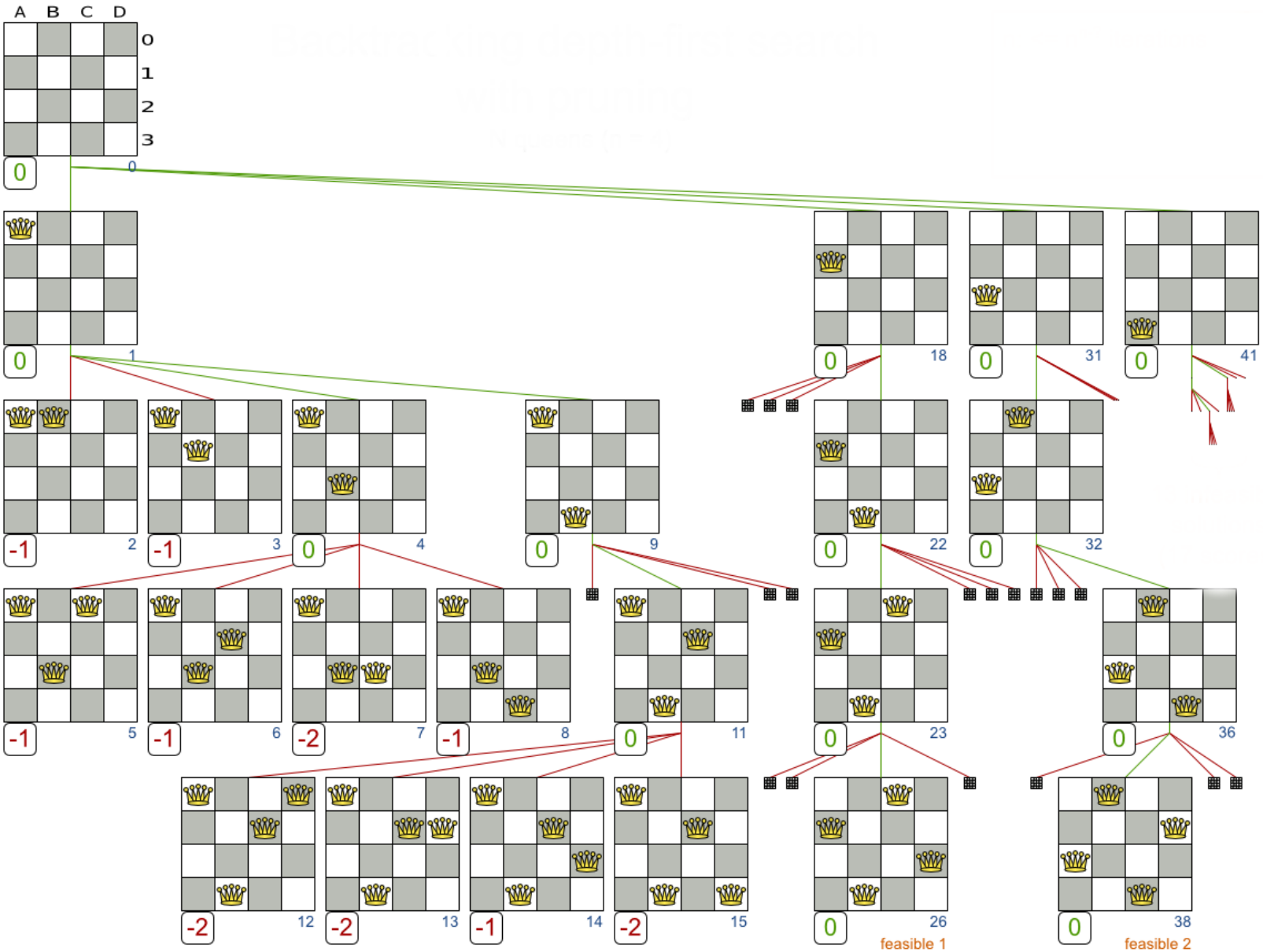
Filtrar vs. Gerar

Duas possíveis abordagens podem ser escolhidas quando fazendo uma busca exaustiva:

- Filtragem – Todas as possíveis soluções são geradas e depois examinadas para eliminar as inválidas.
- Geração – Soluções construídas progressivamente, assim que uma inconsistência é detectada a solução é descartada.

- Candidato parcial (estado) – parte de uma possível solução;
  - Pode ser completado (através de um passo de extensão) de diferentes maneiras para gerar uma solução.
- Candidatos são os nós de uma árvore. Cada candidato parcial é pai dos candidatos que podem ser obtidos a partir dele em um passo de extensão.

Complete Search



Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**
- Obs_1.: `Cada coluna deve ter exatamente uma rainha..`

Complete Search

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**
- Obs_1.: `Cada coluna deve ter exatamente uma rainha..`
- 2. Gerar as n^n possíveis soluções e verificar se são válidas.

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**
- Obs_1.: `Cada coluna deve ter exatamente uma rainha..`
- 2. Gerar as n^n possíveis soluções e verificar se são válidas.
  Melhorou, só que não!

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**
- Obs_1.: `Cada coluna deve ter exatamente uma rainha..`
- 2. Gerar as n^n possíveis soluções e verificar se são válidas.
  Melhorou, só que não!
- Obs_2.: `Cada linha também tem exatamente uma rainha..`

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**
- Obs_1.: `Cada coluna deve ter exatamente uma rainha..`
- 2. Gerar as n^n possíveis soluções e verificar se são válidas.
  Melhorou, só que não!
- Obs_2.: `Cada linha também tem exatamente uma rainha..`
- 3. Gerar as $n!$ possíveis soluções e verificar se são válidas.

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Abordagens..

- 1. Gerar n pares (x, y) e verificar se formam uma solução válida.
  n^{2n} – **BAD!**
- Obs_1.: `Cada coluna deve ter exatamente uma rainha..`
- 2. Gerar as n^n possíveis soluções e verificar se são válidas.
  Melhorou, só que não!
- Obs_2.: `Cada linha também tem exatamente uma rainha..`
- 3. Gerar as $n!$ possíveis soluções e verificar se são válidas.
  Hmm, parece "bom".. será que dá pra melhorar?

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez $n \times n$ de modo que uma rainha não ataque outra.

Busca em profundidade, Backtracking

- Podemos tentar adicionar as rainhas uma por uma, recursivamente, no tabuleiro.

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez $n \times n$ de modo que uma rainha não ataque outra.

Busca em profundidade, Backtracking

- Podemos tentar adicionar as rainhas uma por uma, recursivamente, no tabuleiro.
- Explorando o fato de que é necessário colocar uma rainha por coluna, em cada passo da recursão basta escolher em qual linha na coluna atual colocar a rainha.

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez n x n de modo que uma rainha não ataque outra.

Busca em profundidade, Backtracking

- Podemos tentar adicionar as rainhas uma por uma, recursivamente, no tabuleiro.
- Explorando o fato de que é necessário colocar uma rainha por coluna, em cada passo da recursão basta escolher em qual linha na coluna atual colocar a rainha.
- Não faz sentido colocar uma rainha em uma posição que entra na zona de ataque de alguma das rainhas anteriormente colocadas.

Complete Search

Problema: n Queens

Colocar n rainhas em um tabuleiro de xadrez $n \times n$ de modo que uma rainha não ataque outra.

Busca em profundidade, Backtracking

- Podemos tentar adicionar as rainhas uma por uma, recursivamente, no tabuleiro.
- Explorando o fato de que é necessário colocar uma rainha por coluna, em cada passo da recursão basta escolher em qual linha na coluna atual colocar a rainha.
- Não faz sentido colocar uma rainha em uma posição que entra na zona de ataque de alguma das rainhas anteriormente colocadas.
- Continuamos tentando gerar as $n!$ permutações, mas agora só geramos aquelas que são válidas. (filtrar vs. gerar)

Problema: *n* Queens

Colocar *n* rainhas em um tabuleiro de xadrez *n* x *n* de modo que uma rainha não ataque outra.

Busca em profundidade, Backtracking

```
void backtrack(int col) {
    if (col == n) { /* do something.. */ return; }
    for (int row = 0; row < n; ++row) {
        if (!under_attack[row][col]) {
            placeQueen(row, col);
            backtrack(col + 1);
            removeQueen(row, col);
        }
    }
}
```

Busca em profundidade, Backtracking

- Essa abordagem é um exemplo de uma busca em profundidade (DFS - *Depth First Search*)
  - O algoritmo tenta iterar do topo ao fundo da árvore o mais rápido possível.
  - Um vez que k rainhas são colocadas no tabuleiro, apenas tabuleiros com mais rainhas são examinados.
- O algoritmo busca na árvore de cima para baixo, analisando os candidatos parciais.
- Quando uma solução é encontrada ou uma inconsistência detectada, o mecanismo de *backtracking* entra em ação
  - O caminho que levou até aquela solução é percorrido ao contrário até que uma nova extensão possa ser gerada.

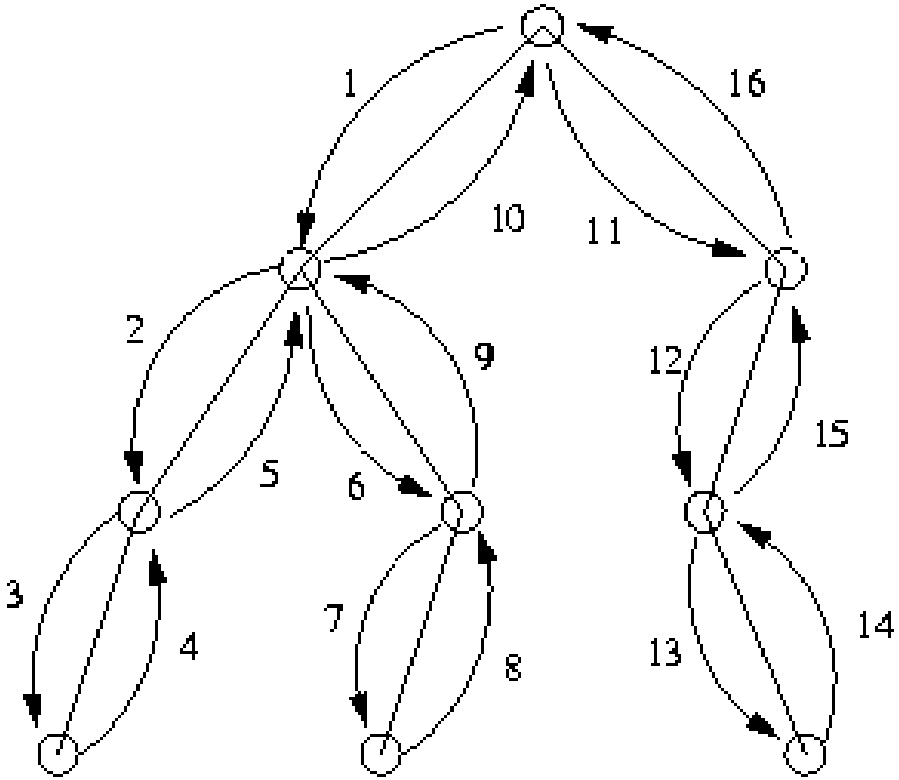


Figura : USACO

Busca em profundidade, Backtracking - Complexidade

- Seja d o número de decisões que devem ser feitas

  No caso das n-rainhas $d = n$, o nro. de colunas que devemos preencher.
- Seja C a quantidade de escolhas para cada decisão

  No caso das n-rainhas $C = n$, já que qualquer uma das linhas pode ser escolhida.
- No pior caso, a busca leva tempo $O(C^d)$, ou seja, uma quantidade exponencial de tempo.
- Entretanto, a quantidade de espaço necessária é bem pequena.
  - Como só é necessário manter informação das decisões a serem feitas, apenas $O(d)$ espaço é necessário.

Problema: Knight Cover

Colocar o menor número de cavalos em um tabuleiro de xadrez $n \times n$ de modo que toda célula do tabuleiro está sob ataque. **Um cavalo não ataca a posição onde ele se encontra.*

Problema: Knight Cover

Colocar o menor número de cavalos em um tabuleiro de xadrez $n \times n$ de modo que toda célula do tabuleiro está sob ataque. **Um cavalo não ataca a posição onde ele se encontra.*

Busca em Largura

- Como queremos o menor número de cavalos, é mais interessante examinar todas as soluções com k cavalos antes de partir para aquelas com $k + 1$.

Problema: Knight Cover

Colocar o menor número de cavalos em um tabuleiro de xadrez $n \times n$ de modo que toda célula do tabuleiro está sob ataque. **Um cavalo não ataca a posição onde ele se encontra.*

Busca em Largura

- Como queremos o menor número de cavalos, é mais interessante examinar todas as soluções com k cavalos antes de partir para aquelas com $k + 1$.
- Essa abordagem é um exemplo de uma busca em largura (BFS - *Breadth First Search*)

Problema: Knight Cover

Colocar o menor número de cavalos em um tabuleiro de xadrez $n \times n$ de modo que toda célula do tabuleiro está sob ataque. **Um cavalo não ataca a posição onde ele se encontra.*

Busca em Largura

- Como queremos o menor número de cavalos, é mais interessante examinar todas as soluções com k cavalos antes de partir para aquelas com $k + 1$.
- Essa abordagem é um exemplo de uma busca em largura (BFS - *Breadth First Search*)
- Geralmente a implementação envolve uma fila de estados / candidatos parciais

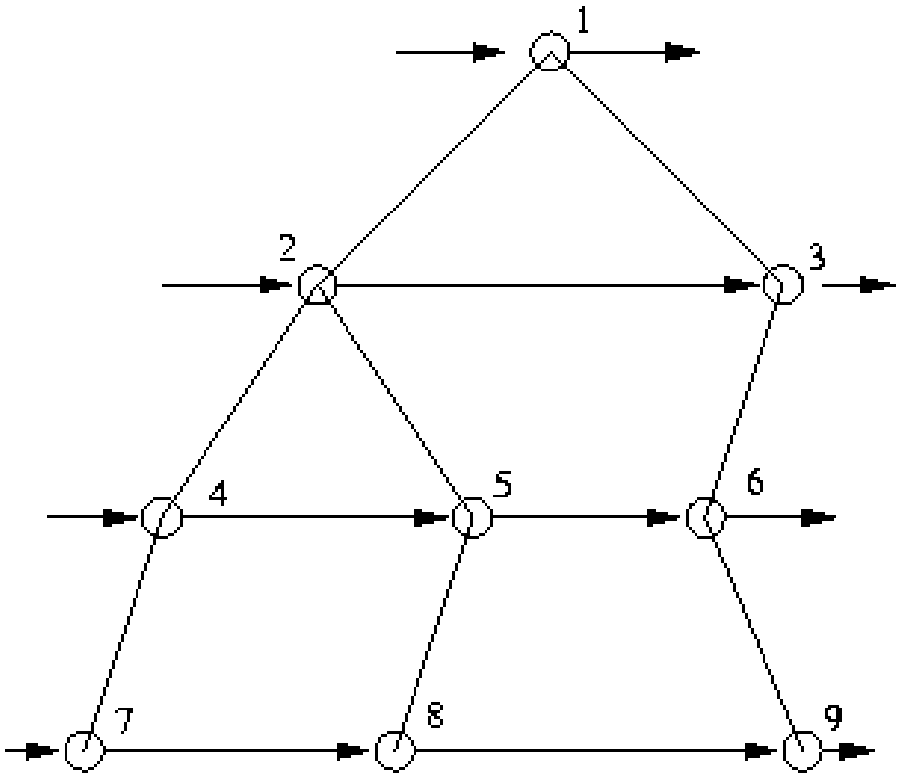


Figura : USACO

Busca em Largura

```
struct state_t { /* ... */ };

void search() {
    queue< state_t > q;
    q.push(/* initial state */);
    int level = 0;
    while (!q.empty()) {
        int candidates_at_lvl = q.size();
        while (candidates_at_lvl--) {
            state_t cur = q.front(); q.pop();
            /* if 'cur' is what the problem asks,
                'level' is the minimum number of needed steps. */

            /* for each possible 'nxt' state from this one */
            q.push(nxt);
        }
        level++;
    }
}
```

Busca em largura

- Chamada de busca em largura porque percorre uma linha inteira (a largura) da árvore de candidatos antes de passar para a próxima linha.
- Primeiro visita a raiz, então todos os nós no nível 1, depois todos no nível 2, ...

Busca em largura

- Chamada de busca em largura porque percorre uma linha inteira (a largura) da árvore de candidatos antes de passar para a próxima linha.
- Primeiro visita a raiz, então todos os nós no nível 1, depois todos no nível 2, ...

Busca em largura – Complexidade

- Complexidade de tempo é a mesma da busca em profundidade.
- Consumo de espaço proporcional ao número de candidatos.
  - Sejam c o número de escolhas para cada decisão, e k o número de decisões que devem ser feitas.
  - Existem c^k possíveis candidatos que vão estar na fila para o próximo passo.

Busca em aprofundamento iterativo

- *Depth First with Iterative Deepening* (ID)
- Alternativa a busca em largura
- São executadas sequencialmente D buscas em profundidade
- Cada busca pode ir um nível além da busca anterior
- Simula uma busca em largura, complexidade de tempo pior mas gasta menos espaço

Busca em aprofundamento iterativo

```
struct state_t{ /* ... */ };

void truncated_dfs(state_t cur, int depth) {
    /* if 'cur' is what the problem asks, we've found a solution */

    if (depth == 0) return;

    /* for each possible 'nxt' state from this one */
    truncated_dfs(nxt, depth-1);
}

void dfid_search() {
    for (int depth = 0; depth <= MAX_DEPTH; ++depth)
        truncated_dfs(/* initial state */, depth);
}
```

Busca em aprofundamento iterativo – Complexidade

- Complexidade de espaço é a mesma da busca em profundidade;
- Complexidade de tempo pior:
  - Busca em profundidade parando na profundidade k leva $O(c^k)$
  - Seja d o número máximo de decisões (profundidade máxima), o tempo gasto será $c^0 + c^1 + c^2 + c^3 + ... + c^d$
- Sempre que há pelo menos duas escolhas a serem tomadas, a busca em aprofundamento iterativo não gasta mais que o dobro de tempo que a busca em largura teria gasto.

Quando usar?

Busca	Tempo	Espaço	Quando usar
Profundidade	$O(c^k)$	$O(k)$	a) De qualquer maneira vai olhar todos os estados, b) sabe o nível que a resposta está, ou c) não está procurando a menor resposta.
Largura	$O(c^d)$	$O(c^d)$	a) Sabe que a resposta fica perto do topo da árvore, ou b) está procurando a menor resposta.
Aprofundamento Iterativo	$O(c^d)$	$O(d)$	Quer fazer uma busca em largura, não tem espaço e pode gastar um pouco mais de tempo.

Hora de resolver alguns problemas..

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.
  - Pior caso: $O(9^k)$, onde k é o menor nro. de movimentos necessário.

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.
  - Pior caso: $O(9^k)$, onde k é o menor nro. de movimentos necessário.
- obs_1: `ordem dos movimentos não importa`

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.
  - Pior caso: $O(9^k)$, onde k é o menor nro. de movimentos necessário.
- obs_1: `ordem dos movimentos não importa`
- Abordagem 2 – para cada movimento, variar a quantidade de vezes que ele será realizado.

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.
  - Pior caso: $O(9^k)$, onde k é o menor nro. de movimentos necessário.
- obs_1: `ordem dos movimentos não importa`
- Abordagem 2 – para cada movimento, variar a quantidade de vezes que ele será realizado.
  - Complexidade: $O(k^9)$

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.
  - Pior caso: $O(9^k)$, onde k é o menor nro. de movimentos necessário.
- obs_1: `ordem dos movimentos não importa`
- Abordagem 2 – para cada movimento, variar a quantidade de vezes que ele será realizado.
  - Complexidade: $O(k^9)$
- obs_2: `fazer um movimento 4 vezes é o mesmo que não fazer nenhuma`

The Clocks [IOI 94]

- Existem 9 relógios em um *grid* 3 x 3; que podem estar mostrando um dos seguintes horários: 12:00, 3:00, 6:00 ou 9:00.
- Objetivo: Fazer com que todos mostrem 12:00.
- 9 comandos podem ser usados para manipular os relógios
  - Cada comando rotaciona um subconjunto de relógios 90 graus no sentido horário.
- Qual a menor sequência de comandos que faz com que todos os relógios mostrem 12:00.

- Abordagem 1 – incrementar a quantidade de movimentos que podem ser usados e tentar encontrar uma sequência com essa quantidade de movimentos recursivamente.
  - Pior caso: $O(9^k)$, onde k é o menor nro. de movimentos necessário.
- obs_1: `ordem dos movimentos não importa`
- Abordagem 2 – para cada movimento, variar a quantidade de vezes que ele será realizado.
  - Complexidade: $O(k^9)$
- obs_2: `fazer um movimento 4 vezes é o mesmo que não fazer nenhuma`
- Basta testar $4^9 = 262144$ possibilidades!

UVa 725 – Division

Encontrar dois números de 5 dígitos tais que, **abcde**/**fghij** $= N$. Cada digito de 0-9 deve aparecer exatamente uma vez, $2 \leq N \leq 79$.

UVa 725 – Division

Encontrar dois números de 5 dígitos tais que, **abcde**/**fghij** $= N$. Cada digito de 0-9 deve aparecer exatamente uma vez, $2 \leq N \leq 79$.

- Testar todos os valores para **fghij**
- **abcde** $=$ **fghij** $* N$
- Verifica restrição de uso dos dígitos

Superprime Rib [USACO 1994 Final Round, adapted]

Um número X é dito superprimo se X é primo e todo número obtido apagando alguma quantidade de dígitos à direita da representação decimal de X é primo. Por exemplo, 233 é superprimo, já que 233, 23 e 2 são todos primos.

- Imprima uma lista de todos os superprimos de tamanho n, ($n \leq 9$). 1 não é primo.

Superprime Rib [USACO 1994 Final Round, adapted]

Um número X é dito superprimo se X é primo e todo número obtido apagando alguma quantidade de dígitos à direita da representação decimal de X é primo. Por exemplo, 233 é superprimo, já que 233, 23 e 2 são todos primos.

- Imprima uma lista de todos os superprimos de tamanho n, ($n \leq 9$). 1 não é primo.

- Construir o número dígito por dígito.

Superprime Rib [USACO 1994 Final Round, adapted]

Um número X é dito superprimo se X é primo e todo número obtido apagando alguma quantidade de dígitos à direita da representação decimal de X é primo. Por exemplo, 233 é superprimo, já que 233, 23 e 2 são todos primos.

- Imprima uma lista de todos os superprimos de tamanho n, $(n \leq 9)$. 1 não é primo.

- Construir o número dígito por dígito.
- Todas as repostas estão a uma profundidade n

Superprime Rib [USACO 1994 Final Round, adapted]

Um número X é dito superprimo se X é primo e todo número obtido apagando alguma quantidade de dígitos à direita da representação decimal de X é primo. Por exemplo, 233 é superprimo, já que 233, 23 e 2 são todos primos.

- Imprima uma lista de todos os superprimos de tamanho n, ($n \leq 9$). 1 não é primo.

- Construir o número dígito por dígito.
- Todas as repostas estão a uma profundidade n
- Busca em profundidade!

UVa 11742 – Social Constraints

- $0 < n \leq 8$ amigos vão ao cinema
- Vão se sentar na primeira fileira, com n assentos consecutivos
- Existem $0 \leq m \leq 20$ restrições, (a, b, c) indicando que a e b devem estar sentados no máximo a c assentos de distancia
- De quantas formas eles podem se sentar?

UVa 11742 – Social Constraints

- $0 < n \leq 8$ amigos vão ao cinema
- Vão se sentar na primeira fileira, com n assentos consecutivos
- Existem $0 \leq m \leq 20$ restrições, (a, b, c) indicando que a e b devem estar sentados no máximo a c assentos de distancia
- De quantas formas eles podem se sentar?

- Testar as $n!$ permutações `/* C++ - next_permutation() */`
- Verifica se a permutação satisfaz todas as restrições

UVa 12346 - Water Gate Management

Uma barragem tem $1 \leq n \leq 20$ portões que deixam a água passar quando necessário, cada portão tem uma taxa que determina quanta água ele é capaz de deixar passar por segundo e um custo de abertura.

- Sua tarefa é controlar a abertura dos portões de modo que a agua possa fluir a uma taxa de x unidades por segundo e o custo total de abertura seja minimo.

UVa 12346 - Water Gate Management

Uma barragem tem $1 \leq n \leq 20$ portões que deixam a água passar quando necessário, cada portão tem uma taxa que determina quanta água ele é capaz de deixar passar por segundo e um custo de abertura.

- Sua tarefa é controlar a abertura dos portões de modo que a agua possa fluir a uma taxa de x unidades por segundo e o custo total de abertura seja minimo.

- Testar todos os 2^n subconjuntos de portões que podem ser abertos
- Para cada subconjunto:
  - Verifica se a taxa passando é maior ou igual a taxa desejada
  - em caso positivo, verifique se o custo é menor que o menor custo encontrado ate agora

Party Lamps [IOI 98]

Existem N lâmpadas numeradas de 1 a N e 4 interruptores. O primeiro interruptor alterna todas as lâmpadas, o segundo as lâmpadas pares, o terceiro as lâmpadas impares, e o último as lâmpadas 1, 4, 7, 10, ...

- Dados o número de lâmpadas N, o número de vezes que algum interruptor foi pressionado (no máximo 10000), e o estado de algumas lâmpadas (ex., lâmpada 7 está desligada), imprima todas as configurações em que as lâmpadas podem estar.

Party Lamps [IOI 98]

Existem N lâmpadas numeradas de 1 a N e 4 interruptores. O primeiro interruptor alterna todas as lâmpadas, o segundo as lâmpadas pares, o terceiro as lâmpadas impares, e o último as lâmpadas 1, 4, 7, 10, ...

- Dados o número de lâmpadas N, o número de vezes que algum interruptor foi pressionado (no máximo 10000), e o estado de algumas lâmpadas (ex., lâmpada 7 está desligada), imprima todas as configurações em que as lâmpadas podem estar.

- Para cada botão pressionado: 4 possibilidades. 4^{10000}. **TLE!**
- obs_1: `Ordem que os interruptores são pressionados não importa.`
- obs_2: `Pressionar um interruptor 2 vezes é o mesmo que não pressionar nenhuma.`
- Só precisamos testar se um botão foi ou não apertado, $2^4 = 16$ possibilidades.

Addition Chains

Uma "cadeia de adição" é uma sequência de inteiros tal que o primeiro número é 1, e todo número subsequente é a soma de dois termos que aparecem na sequência antes dele.
Por exemplo, 1 2 3 4 é uma cadeia de adição uma vez que, $2 = 1 + 1$, $3 = 2 + 1$ e $5 = 2 + 3$.

- Encontre o tamanho da menor cadeia que termina com um dado número.

Addition Chains

Uma "cadeia de adição" é uma sequência de inteiros tal que o primeiro número é 1, e todo número subsequente é a soma de dois termos que aparecem na sequência antes dele.
Por exemplo, 1 2 3 4 é uma cadeia de adição uma vez que, $2 = 1 + 1$, $3 = 2 + 1$ e $5 = 2 + 3$.

- Encontre o tamanho da menor cadeia que termina com um dado número.

- Busca em aprofundamento iterativo é uma boa alternativa:
  - A tendência é que a busca em profundidade gere primeiro soluções do tipo **1 2 3 4 5 ... n**
  - A fila na busca em largura cresce muito rapidamente.

Complete Search

Dicas

- Podar o quanto antes

Complete Search

Dicas

- Podar o quanto antes
- Aproveitar simetrias

Complete Search

Dicas

- Podar o quanto antes
- Aproveitar simetrias
- Pré-cálculo

Complete Search

Dicas

- Podar o quanto antes
- Aproveitar simetrias
- Pré-cálculo
- Otimizar código

Complete Search

Dicas

- Podar o quanto antes
- Aproveitar simetrias
- Pré-cálculo
- Otimizar código
- Usar um algoritmo/estrutura de dados melhor

Leituras Recomendadas

- http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=recursionPt1
- http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=recursionPt2
- http://www.inf.ufg.br/~paulocosta/tap/material/bt1.pdf
- http://www.inf.ufg.br/~paulocosta/tap/material/bt2.pdf
- http://www.comp.nus.edu.sg/~stevenha/visualization/recursion.html