**Greedy Search**

Chamamos algoritmos gananciosos quando eles utilizam a propriedade ganancioso. A Greedy Search é:

Nesse exato momento, qual é a melhor escolha a ser feita?

Algoritmos gananciosos são gananciosos. Eles não olham para o futuro para decidir a solução ideal global. Eles estão preocupados apenas com a solução ideal localmente. Isso significa que a solução ideal geral pode diferir da solução escolhida pelo algoritmo.

Eles nunca olham para trás para o que fizeram para ver se poderiam otimizar globalmente. Essa é a principal diferença entre Greedy e Dynamic Programming .

Para ser mais claro, uma das perguntas mais pesquisadas no Google sobre algoritmos gananciosos é:

"Quais estratégias de solução de problemas não garantem soluções, mas fazem uso eficiente do tempo?"

A resposta é "algoritmos gananciosos". Eles não garantem soluções, mas são muito eficientes em termos de tempo. No entanto, na próxima seção, aprenderemos que às vezes as soluções gananciosas nos fornecem as soluções ideais.

É ideal localmente, mas às vezes não é ideal globalmente. No algoritmo de mudança, podemos forçar um ponto em que não é ideal globalmente.

O algoritmo para fazer isso é:

Escolha 3 denominações de moedas. 1p, x, e menos que 2x, mas mais que x.

Vamos escolher 1, 15, 25.

Peça mudança de 2 \* segunda denominação (15)

Pediremos a alteração de 30. Agora, vamos ver o que nosso algoritmo Greedy faz.

**A\* Search**

O algoritmo de A\* Search é uma extensão do algoritmo de Dijkstra, útil para encontrar o caminho de menor custo entre dois nós (também conhecidos como vértices) de um gráfico. O caminho pode atravessar qualquer número de nós conectados por arestas (aka arcos), com cada aresta tendo um custo associado. O algoritmo usa uma heurística que associa uma estimativa do caminho de custo mais baixo desse nó ao nó de objetivo, de modo que essa estimativa nunca seja maior que o custo real.

O algoritmo não deve assumir que todos os custos de borda são os mesmos. Deve ser possível iniciar e concluir qualquer nó, incluindo aqueles identificados como uma barreira na tarefa.

Considere o problema de encontrar uma rota através da diagonal de uma grade 8x8 semelhante a um tabuleiro de xadrez. As linhas são numeradas de 0 a 7. As colunas também são numeradas de 0 a 7. A posição inicial é (0, 0) e a posição final é (7, 7). O movimento é permitido por um quadrado em qualquer direção, incluindo diagonais, semelhantes a um rei no xadrez. O custo de movimento padrão é 1. Para tornar as coisas um pouco mais difíceis, existe uma barreira que ocupa certas posições da grade. Mover para qualquer uma das posições de barreira tem um custo de 100.

A barreira ocupa as posições (2,4), (2,5), (2,6), (3,6), (4,6), (5,6), (5,5), (5, 4), (5,3), (5,2), (4,2) e (3,2).

Uma rota com o menor custo deve ser encontrada usando o algoritmo de busca A \* (existem várias soluções ideais com o mesmo custo total).

Imprima a rota ideal em formato de texto, bem como o custo total da rota.

Opcionalmente, desenhe a rota ideal e as posições da barreira.

Nota: usar uma pontuação heurística igual a zero é equivalente ao algoritmo de Dijkstra e isso é meio que trapaça / não é realmente A \*!

**Graph Search**

Graph são redes constituídas por nós conectados por arestas ou arcos. Nos gráficos direcionados, as conexões entre os nós têm uma direção e são chamadas de arcos; em gráficos não direcionados, as conexões não têm direção e são chamadas de arestas. Discutimos principalmente gráficos direcionados. Os algoritmos nos gráficos incluem encontrar um caminho entre dois nós, encontrar o caminho mais curto entre dois nós, determinar ciclos no gráfico (um ciclo é um caminho não vazio de um nó para si), encontrar um caminho que alcance todos os nós (o famoso "problema do vendedor ambulante") e assim por diante. Às vezes, os nós ou arcos de um gráfico têm pesos ou custos associados a eles, e estamos interessados ​​em encontrar o caminho mais barato.

Há literatura considerável sobre algoritmos de gráficos, que são uma parte importante da matemática discreta. Os gráficos também têm muito uso prático em algoritmos de computador. Exemplos óbvios podem ser encontrados no gerenciamento de redes, mas existem muitos exemplos em muitas outras áreas. Por exemplo, os relacionamentos de chamador-chamado em um programa de computador podem ser vistos como um gráfico (onde os ciclos indicam recursão e os nós inacessíveis representam o código morto).