Resposta da Tarefa - trab 2

Teoria da computação

Luiz Antonio Roque Guzzo

PPComp — Campus Serra, Ifes

December 2022

1 Solução do problema:

Nome do exercício: Nintai
Figura 1 tem o 'Regex-Crossword' sem a resposta. E a figura 2 tem a resposta do 'Regex-Crossword'.

2 Solução do problema:

Peguei uma implementação razoável do github que o autor utilizou o algoritmo de Thompson para criar as AFND que tenta implementar o funcionamento da engine do RegEx, tentei fazer algumas alterações para sanar a necessidade do trabalho. Leia o Read-me correspondente para mais detalhes.

3 Solução do problema:

3.1

Para demonstrar que o problema de "Dominating Set"é um problema NP-completo, é necessário mostrar que ele pode ser resolvido de forma eficiente sendo reduzida a outros problemas NP-completos. Uma das formas de fazer isso é através da redução do problema de "Vertex-Cover"que foi previamente resolvido por Karp como NP-completo e foi abordado pelo Teorema 7.44 do livro de Sipser e na lista dos 21 problemas NP-Completos (17.3) do livro "What Can Be Computed".

A redução é feita através da seguinte sequência: Vertex-Cover > Dominating Set > Simplified Dominating Set, onde cada passo da redução garante que sejam mantidos os

Figura 1: Problema 1

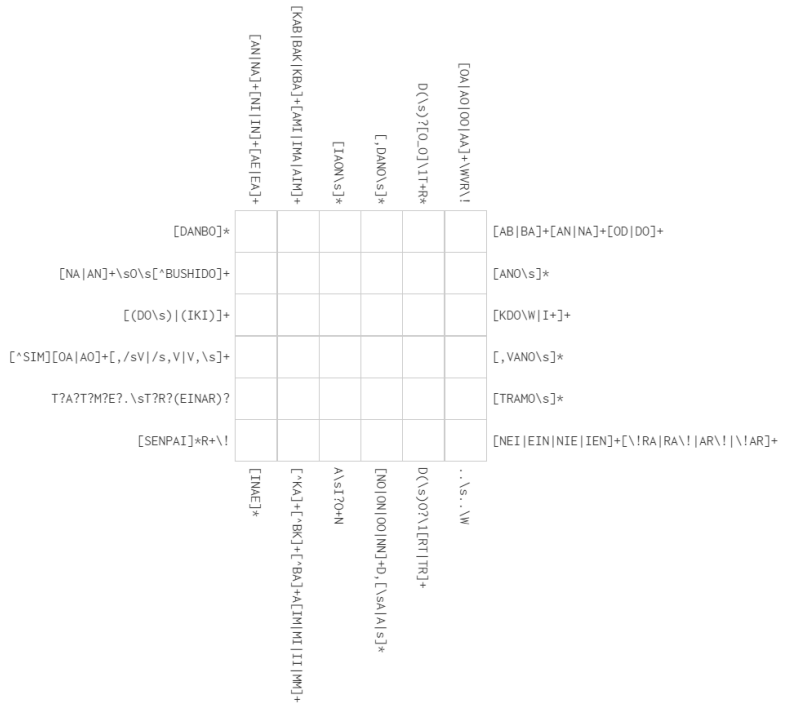
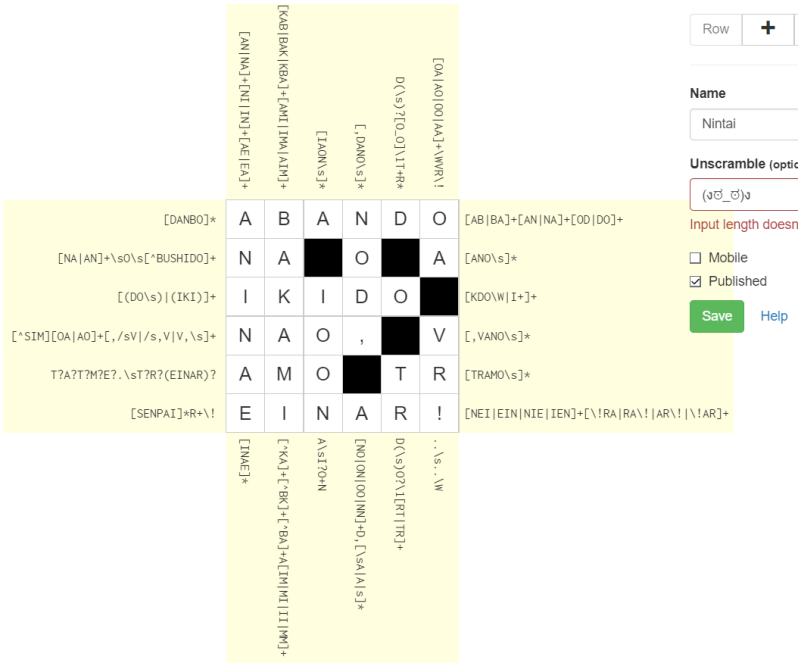


Figura 2: Problema 1



aspectos NP-completos do problema, permitindo assim a demonstração de que o problema de "Dominating Set"é um problema NP-completo.

Passo a passo da redução:
(referencia da redução: https://www.youtube.com/watch?v=HueFKEg7QZ8):

1. Converte o grafo original G em um grafo bipartido H:
   a) Se faz 2 conjuntos A e B como cópia dos vértices do grafo G.
   b) Coloca-se uma aresta entre cada vértice de A para B se eles forem adjacentes a G
2. Calcula o grau de todos os vértices de A
3. a) Seleciona o vértices de A com o maior grau e adiciona o vértice V ao um conjunto D
   b) Remova todos os vértices no conjunto B que estão cobertos pelo vértice eleito V, inclusive ele mesmo
   c) Remova o vértice V de A
4. Repete o passo 2 depois o 3 até que não haja mais vértices no conjunto B, os vértices contidos no conjunto D é o conjunto dominante

Após o processo de redução de Vertex-Cover para Dominating Set, agora é a vez do Dominating Set para sua versão simplificada que consiste em printar "Sim"caso haja Conjunto Dominante e NIL caso não tenha.

O único cenário que retornaria NIL seria quando o grafo não possuir conectividade entre seus nós. Devido a expressão dada pelo exercício que diz:

Para todo u, se u pertence ao conjunto de vértices (V) do conjunto dominante, então, existe algum v que pertence ao conjunto dominante (D) e existe alguma aresta u,v que pertence ao conjunto de arestas E.

Se não existe alguma aresta que conecta a um vértice então não existe um conjunto Dominante para este grafo. (referencia: video de duvidas de TC)

3.2

Está no codigo, leia o READ-ME correspondente para mais detalhes

4 Solução do problema:

4.1

A fim de provar que o problema de recrutramento (EMP) é NP-completo, preciso mostrar que ele pode ser transformado em outro problema NP-completo conhecido, irei utilizar

o "subset-sum"que é o problema de subconjunto que também é NP-completo (teorema 7.56 do livro de Sipser)

O problema do subconjunto consiste-se em, dado um conjunto de numeros X1,...Xk e um numero alvo inteiro t. Queremos determinar se a coleção contém uma subcoleção que soma t. O problema é NP-completo porque não há uma solução ótima para ele, logo é necessário verificar todas as possibilidades.

Para transformar o problema do recrutamento no problema do subconjunto, basta considerar cada competência como um número inteiro e o conjunto de habilidades de cada candidato como um subconjunto de X. O objetivo é encontrar um subconjunto de candidatos que cobre todas as habilidades de X (ou seja, que soma t = m).

Como o problema do subconjunto é NP-completo, e o problema do recrutamento pode ser transformado nele, então podemos concluir que o problema do recrutamento também é NP-completo.

4.2

Codigo da primeira versão, encontra-se na pasta correspondente

4.3

Em vez de realizar um loop que testa todas as possibilidades de equipes oque configuraria O(N**k) pois o algoritmo gera todas as combinações possíveis de tamanho k para uma lista de tamanho n. Havia pensado em realizar loops mais precisos para verificar cada membro da equipe montada com cada candidato, a fim de buscar o candidato que cresce o tamanho de competências da equipe, com isso fiz a nova versão que busca construir uma equipe e a cada interação um membro é trocado até que se monte uma equipe que atenda todas as competências.

Esta nova versão de código tenta resolver o problema de selecionar uma equipe de trabalho, ele começa selecionando um K membros iniciais para a equipe e os demais não selecionados são considerados como pendentes. Em seguida, ele realiza loops para verificar cada candidato dos pendentes e escolher o melhor para a equipe. Ele faz isso comparando cada candidato dos pendentes com cada membro da equipe e verificando se a adição desse candidato melhora a equipe. Caso seja encontrado um melhor candidato, ele troca com o pior membro da equipe, caso contrário, ele pega o primeiro da fila de pendentes e do time. Ele faz essa verificação para todos os candidatos, o que significa que ele roda len(L) vezes, e dentro desse loop há outros loops para verificar cada pendente (len(L)-k), comparar cada membro do pendente com cada membro da equipe (k), verificar se todas as competências foram verificadas (k), e após identificar a equipe competente, ele roda um loop para converter a resposta em um array de index (k). Ou seja, as operações resumidamente seriam:

1. A função EMP roda len(L) vezes, pegando todos os membros;
2. O loop interno do EMP roda len(L)-k vezes, para cada novo candidato;

3. Para cada membro da equipe selecionada é realizado um loop, roda K vezes;

4. A função competências-cobertas roda mais k vezes dentro para verificar a permutação de cada membro com o novo candidato, se houve uma melhoria nas competências atendidas;

5. A função equipe-to-index roda k vezes, e é realizada apenas 1 vez para conversão do resultado;

Juntando os loops o algoritmo seria algo próximo de O(L* (L-k) * k**2 + k), portanto a complexidade do código seria Polinomial, sendo superior após um certo tamanho da equipe solicitada em comparação versão anterior em que o tempo de complexidade é O(L**k)

Informações de como executar a implementação encontra-se no READ-me.

5 Solução do problema:

5.1

Acabei fazendo 2 questões Medias, uma que não conseguiu passar em todos os testes devido a lentidão (time limit exceeded), e outra que consegui fazer totalmente.

A incompleta: https://open.kattis.com/problems/paintball
A completa:
Os devidos codigos se encontram nas pastas correspondentes, leia o READ-me caso tenha duvidas

5.2

Informações de como executar a implementação encontra-se no READ-me.