CES12 - Lab 2020
Paradigmas de Programação
SSP - Subset Sum Problem

Versão 200609

- Não muda nenhuma funcionalidade ou interface para o aluno
- Apenas atualiza header obsoleto e chamadas internas a funções não-standard que podem (depende do seu sistema) **facilitar a compilação**. Só isso.
- Se conseguiu compilar com a versão anterior, não é necessário mudar. Provavelmente a geração de instâncias com distribuição diferente não fará diferença na prática.
- Os erros que podem ser corrigidos por esta versão são:
    - Falta do header `<values.h>`.
    - Falta da função `random()`.

Mudanças (nenhuma afeta o código do aluno)

1. Em `include/subsetsum.h`, o header obsoleto `<values.h>` foi substituído por `<float.h>` (se trata da lib interna com a solução, não interfere no código do aluno)
2. Na classe `SSPInstanceGenerator`, chamadas a função `random()` (que faz parte do standard POSIX mas não do standard C++) foram substituídas pelos geradores de números aleatórios do header `<random>`, parte do standard C++11.
3. O código que gera as instâncias PX e EVENODD agora é como nos slides (usando uma distribuição uniforme, diretamente com a API `<random>`), eliminando o algoritmo de incremento usado antes.

O problema SSP

O problema SSP é uma simplificação do Knapsack (mochila) geral, onde todos os itens tem lucro igual ao peso, e não interessa obter uma soma menor do que o tamanho da mochila: a pergunta é apenas : existe um subconjunto de itens cuja soma de pesos é igual ao tamanho da mochila?

Mesmo o problema simplificado ainda é NP-completo e pseudo-polinomial.

Existem várias soluções elaboradas, e o aluno deve implementar:

- Uma solução por programação dinâmica (PD)
- Uma outra solução, chamada de BB: e.g. Branch-and-Bound (BB) ou Meet-in-the-Middle. Apesar do nome BB, pode ser outra solução, exceto Programação Dinâmica que já foi coberta pela primeira solução.

Implementação

Existe uma classe derivada para PD e outra para BB, fornecidas com métodos em branco a serem implementados pelo aluno. Ambas herdam da seguinte classe abstrata:

```
class SubsetSumSolverAbstract {
public:

      virtual bool solve(const std::vector< long> &input,
                        long value, std::vector< char> &output)=0;

      virtual std::string getName() { return "ABSTRACT"; }

};
```

O método getName já está implementado no arquivo header fornecido para as classes derivadas. Portanto o aluno deverá implementar o método solve para as duas classes derivadas fornecidas.

O método solve recebe como entrada:

- Input vetor de longs, inteiros contendo os pesos dos itens.
- value capacidade da mochila

Long aqui não é exagero, inclusive um dos geradores de instâncias teve que ser limitado para não estourar o tipo long.

Como saída:

- O método deve retornar true se existe solução; e false caso contrário.
- Se o método retorna true, deve preencher, como saída, o vetor output com true/false (é char, mas usamos como bool), onde output[i] = true significa que o item i faz parte da solução encontrada.
    - Se não existe solução e o método retorna false, o vetor output não deve representar uma solução válida, obviamente.

Relatório

Só vale implementar algoritmos que o aluno compreende. Se o aluno implementa e não explica no relatório, pode ser convidado a explicar o algoritmo.

Descrição da implementação

Há inúmeros algoritmos, variantes e melhorias que podem ser aplicados à ambos BB e PD. Descreva qualquer melhoria implementada sobre os algoritmos apresentados na aula teórica e as referências usadas, caso houver. Comente o código mais relevante para as melhorias para permitir relacionar o relatório com o código.

- Cite nominalmente qual o algoritmo escolhido para o seu “BB” (Branch and Bound, Meet in the Middle, ou outro)
- Descreva especialmente as regras de ‘poda’ da árvore de busca do BB (Valem diagramas, fotos de desenhos à mão, etc).
- Descreva qualquer melhoria implementada sobre os algoritmos apresentados na aula teórica e as referências usadas, se houver. Comente o código mais relevante para as melhorias para permitir relacionar o relatório com o código.
- Deixe claro a que método está se referindo, não misture o discurso! Se há algo a dizer sobre os 2 métodos, separe em seções.

Comparação dos resultados

Apresente resultados separados para todos e cada um dos geradores de instâncias.
Discuta e explique os resultados, especialmente as diferenças de desempenho dos algoritmos em cada tipo de instância.

- Os geradores de instâncias PX (P3 P4 P5) são similares exceto pelos valores dos números. Diferenças dependendo dos valores dos pesos devem ser visíveis.
    - Uma boa pergunta para começar a pensar: dada uma entrada aleatória, é mais fácil existir solução com valores pequenos ou com valores grandes?

Formato da saída do teste TimedT

Cada linha de texto tem o formato
`<int>,<float>,<int>,<int>`
Correspondendo a
<input size>, <average execution time>, <number of positive responses>, <number of trials>

Código correspondente:

```
result_dump << s <<","<< sum_times/(float)e.repeat <<","<< count_yes
<<"," << e.repeat << std::endl;
```

Bonus (para quem implementou alguma melhoria):

Os algoritmos mais rápidos da turma obterão bônus desde que:

- O aluno explique porque o seu algoritmo melhorou de desempenho, e quais os casos melhores para o seu algoritmo.
- O aluno inclua nos testes os casos relevantes para demonstrar o desempenho (tanto no teste que efetivamente testa, quanto no teste que gera arquivos de resultados).

Testes

Existe uma função de testes genérica que recebe uma struct com listas representando:

- Nomes dos algoritmos (DD PD BB) - DD é o gabarito
- Tamanhos
- Nomes dos geradores de instâncias
- Número de repetições

A função genérica testará todas as combinações dos valores indicados.
Exemplo:

```
INSTANTIATE_TEST_SUITE_P(TestsManySmallUpto20,
           SSPTest,
          ::testing::Values(SSP_Exp({"DD","PD","BB"},
                           {10,15,20},
                            {"P3","P6","P9","RAND", "EVOD"}
                           ,3)
           ));
```

Estes teste testa os 3 algoritmos com tamanhos 10,15,20, com os 5 geradores indicados. Cada combinação (3 x 3 x 5 combinações) é repetida 3 vezes.

Os testes hardcoded testam até tamanho 15.

Testes gerados pelo aluno:

No final do arquivo de teste há o seguinte código:

```
INSTANTIATE_TEST_SUITE_P(InstanceViewerAluno,
                      InstanceView,
                      ::testing::Values( alunoViewInstance() )
                      );
```

```
INSTANTIATE_TEST_SUITE_P(Tests_Aluno,
                        SSPTest,
                        ::testing::Values( alunoTestSSP1() )
                        );
INSTANTIATE_TEST_SUITE_P(Tests_AlunoTimed1,
                        SSPTimedT,
                        ::testing::Values( alunoTestTimed1() )
                        );
```

1. O primeiro não executa nenhum algoritmo, apenas chama os geradores de instâncias e imprime as instâncias para ajudar o aluno a entender as diferentes instâncias.
2. O segundo chama a rotina de teste genérica que efetivamente testa os resultados contra o gabarito (há 3 versões permitindo incluir 3 testes diferentes)
3. O terceiro não testa nada, apenas mede o tempo de execução e gera arquivos de dados (há duas versões permitindo incluir 2 gerações de arquivos diferentes)

O ponto interessante é que as funções `alunoViewInstance()` `alunoTestTimed1()` e `alunoTestSSP1()` estão incluídas no código fonte do aluno:

```
SSP_Exp alunoTestSSP() {
return SSP_Exp(
      // {"DD","BB","PD"},
       // DD-gabarito BB-branch&bound PD-ProgDinamic
      {"DD"}, // choose algorithms
      {},//{10,15}, // choose sizes. if empty, test does nothing
      // possible instance generators: choose as many as you wish to
test
      {"TODD","P3","P6","P9","RAND", "EVOD","AVIS"}
      // note: TOOD and AVIS are not random, do not need to repeat the
test
      ,3 );  // how many times to repeat the test
}
```

As funções numeradas 1,2,3 são apenas possibilidades de executar vários testes com parâmetros diferentes na mesma execução.

- Para os testes PX é possível chamar de P1 a P9. Nenhum dos algoritmos da aula conseguem resolver P9 em tempo/memória razoáveis, mas se precisar de um teste maior, pode usar.
- Também é possível definir um conjunto de testes pequenos para testar um algoritmo durante o desenvolvimento.

Se a lista de tamanhos está vazia, nada será testado.

Apenas editando estes dados, o aluno pode testar quaisquer combinações de tamanho, algoritmo e gerador de instância que quiser, ou gerar medidas de tempo de execução para as condições que quiser.

Sobre pseudo-polinomial

O tamanho de um problema é definido pelo número de bits necessário para representar a entrada.
Quando dizemos que ordenação é O(nlogn), sabemos que o número de trocas e comparações é O(nlogn). Não depende dos valores dos números, desde que possamos comparar os valores e que comparações e trocas de elementos nos vetores tenham tempo constante.

Já se o SSP é O(Wn) (W é o peso máximo dos itens), qual o tamanho da entrada? Depende de W.

1. O número de bits para representar W é log(W) (log na base 2)
2. portanto o tamanho da entrada é t ~ log(W)
3. Assim W ~ 2^t
4. Logo O(Wn) na verdade significa O((2^t) * n) que não é polinomial.

Por um lado, pseudo-polinomial não é polinomial, o que é importante teoricamente (se O(Wn) fosse polinomial, então este algoritmo implicaria que P = NP). E é importante na prática, pois para valores grandes, o algoritmo é lento.
Por outro lado, para valores pequenos de W, o algoritmo se comporta como polinomial. O que não é relevante teoricamente para classificar o problema, mas é importante na prática, pois para valores pequenos o algoritmo é rápido.

Dicas

em `SSPInstanceGenerator.h` há 2 funções disponíveis para imprimir instâncias do SSP, chamadas `streamSSP`, que podem ajudar a debugar.

Regras

Há 5 coisas a fazer: 3 algoritmos de troco + 2 de SSP. E as respectivas comparações e perguntas.

Precisa fazer no mínimo 2 algoritmos, **e tudo o que conseguir fazer/responder/comparar com esses 2 algoritmos**. Depois na segunda entrega pode completar o mesmo relatório.

Recomendo que terminem o troco (os 3 algoritmos + relatorio) para fechar o assunto. Mas isso é mais do que o mínimo para o 1o deadline. (E talvez o Alonso não cubra todos os paradigmas até a 1a entrega)

Um exemplo do mínimo seria 2 algoritmos para o Troco comparados entre eles, sendo que o relatório será completado na segunda entrega que incluirá o 3o algoritmo.

Se não entregar o mínimo conta atraso como 1pt dia na nota total do lab que vale peso 2, parando em 5 dias.

Rationale: o Alonso demorará algumas semanas para cobrir todos os paradigmas. Na primeira entrega, espero que hajam pelo menos 2 algoritmos já cobertos pelo Alonso.

Entrega: na entrega 1 deve-se realizar um zip de toda pasta src ou apenas dos arquivos do problema do Troco?

Deixe todos os arquivos, zipe o diretório normalmente. Se lembra da demonstração que fiz na primeira aula, apenas indico ao meu script qual é o seu diretório, e ele compila tudo. Se vc deletar arquivos .cpp ou .h não consigo compilar. Como na primeira entrega o lab não está pronto 100%, já espero que nem todos os testes passem.

Nota:

O Lab tem peso 2 no bimestre. 50% cada parte (Troco e SSP). Só haverá mais um outro lab até o fim, portanto vale ⅔ da nota de lab no bimestre.

O aluno pode implementar apenas um entre as soluções Backtracking, Branch-and-Bound (BB) e Meet-in-the-Middle?

Sim. Precisa de **uma outra** solução além de PD. Backtracking e Branch-and-Bound são apenas nomes diferentes para buscas exaustivas, mesmo que espertas o suficiente para evitar percorrer todos os ramos da árvore. MM é outro algoritmo, que não é BB ou Backtracking mas também é válido.

FAQ

O vetor de pesos da mochila dos testes está ordenado?

Acho que ordenei os valores, e também para todos os geradores o item máximo é menor do que a capacidade. Vocês podem checar o código dos testes e dos geradores de instâncias pra conferir. Se falhar por não estar ordenado, basta ordenar.

Versão 190503

Os testes exigiam memória demais, pois o tamanho da tabela é O(nW). W era grande demais pois os valores dos elementos dos vetores de entrada eram grandes demais. Por isso, os testes em geral foram diminuídos de tamanho especialmente no W e também no n (embora o número de testes foi aumentado em alguns casos)

Nada foi mudado no código da lib, no código fornecido em src, ou na lib fechada da solução.

- Os testes PX não estão mais hardcoded, é possível chamar de P1 a P9
- Substituí P3 P6 P9 por P3 P4 P5, diminuindo muito o valor dos elementos mas ainda permitindo verificar as diferenças entre P3 e P5
- Eliminei o teste TODD
- Diminuí o tamanho máximo dos vetores de 20 para 15 (embora os alunos que me mostraram conseguiram executar os testes com tamanho 20 em alguns minutos)
- Tanto o P9 quanto o TODD estão fora do testes mas ainda disponíveis para vocês, se implementarem algo mais sofisticado.