Министерство образования и науки РФ
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Расчётно-графическая работа

Методы разработки алгоритмов

по дисциплине **«Технология программирования»**

Работу выполнил студент
Кладковой Максим Дмитриевич
Группа: 5130904/30005
Руководитель: Червинский Алексей Петрович

Содержание

a. Постановка задачи

Задача о рюкзаке (Knapsack problem) заключается в следующем: имеется рюкзак ограниченной вместимости и множество предметов, каждый из которых имеет вес и ценность. Необходимо выбрать некоторые предметы, чтобы максимизировать общую ценность предметов в рюкзаке, не превышая его вместимость.

Цель задачи:

- Разработать программу, которая будет создавать файлы с данными о рюкзаке и предметах.
- Реализовать алгоритмы решения задачи о рюкзаке: динамическое программирование, метод ветвей и границ, метод полного перебора и метод с использованием обратного отсчёта.

b. Описание программы и совершаемых в ней действий

Программа состоит из следующих компонентов:

1. Основной файл - main.cpp, который управляет вводом команд пользователя и вызовом соответствующих функций.
2. Класс Knapsack - класс, который хранит данные о предметах и рюкзаке и реализует методы решения задачи о рюкзаке.
3. Команды - функции, которые выполняют определённые действия в зависимости от команды пользователя: создание файла с предметами, отображение содержимого файла и решение задачи о рюкзаке различными методами.

Основные действия программы:

- CREATE : Создание файла с заданной вместимостью рюкзака и числом предметов, каждый из которых имеет случайный вес и ценность.
- CREATE : Создание файла с предметами на основе данных, введённых пользователем.
- SHOW : Отображение содержимого файла, включая вместимость рюкзака и предметы с их весами и ценностями.
- SOLVE_DP : Решение задачи о рюкзаке методом динамического программирования.
- SOLVE_BT : Решение задачи о рюкзаке методом обратного отсчёта (backtracking).
- SOLVE_BB : Решение задачи о рюкзаке методом ветвей и границ (branch and bound).
- SOLVE_BF : Решение задачи о рюкзаке методом полного перебора (brute force).

c. Описание тестирования

Для тестирования программы были созданы несколько тестовых файлов с разными наборами предметов и вместимостью рюкзака. Программа тестировалась на корректность выполнения команд и точность решений, полученных различными методами.

Тестовые шаги:

1. Создание файла с предметами:
    - Ввод команды CREATE 50 10 items.txt
    - Проверка наличия файла items.txt с корректным содержимым.
2. Отображение содержимого файла:
    - Ввод команды SHOW items.txt
    - Проверка корректности отображения вместимости и предметов.
3. Решение задачи о рюкзаке:
    - Ввод команд SOLVE_DP items.txt, SOLVE_BT items.txt, SOLVE_BB items.txt, SOLVE_BF items.txt
    - Проверка корректности решений и соответствие ожидаемым результатам.

d. Исходный код программы

function.hpp

```
 1:  #ifndef FUNCTION_HPP
 2:  #define FUNCTION_HPP
 3:
 4:  #include <string>
 5:  #include <vector>
 6:
 7:  namespace kladkovoj
 8:  {
 9:    class Knapsack
10:    {
11:      public:
12:        Knapsack(int numItems, int maxWeight);
13:
14:        void addItem(int weight, int value);
15:        void writeToFile(const std::string& filename) const;
16:
17:        int knapsackDP() const;
18:        int knapsackBacktracking() const;
19:        int knapsackBranchAndBound() const;
20:        int knapsackBruteForce() const;
21:
22:      private:
23:        int maxWeight_;
24:        int numItems_;
25:        std::vector<int> weights_;
26:        std::vector<int> values_;
27:    };
28:  }
29:
30:  #endif
```

function.cpp

```
31:   #include <fstream> // ofstream, ifstream
32:   #include <algorithm> // max
33:   #include <functional> // function
34:
35:   #include "function.hpp"
36:
37:   using namespace kladkovoj;
38:
39:   Knapsack::Knapsack(int numItems, int maxWeight)
40:   : maxWeight_(maxWeight), numItems_(numItems)
41:   {}
42:
43:   void Knapsack::addItem(int weight, int value)
44:   {
45:     weights_.push_back(weight);
46:     values_.push_back(value);
47:   }
48:
49:   void Knapsack::writeToFile(const std::string& filename) const
50:   {
51:     std::ofstream file(filename);
52:     if (!file.is_open())
53:       throw std::invalid_argument("Error opening file for writing.");
54:
55:     file << maxWeight_ << ' ' << numItems_ << '\n';
56:     for (size_t i = 0; i < weights_.size(); ++i)
57:       file << weights_[i] << ' ' << values_[i] << '\n';
58:
59:     file.close();
60:   }
61:
62:   int Knapsack::knapsackDP() const
63:   {
64:     std::vector<std::vector<int>> dp(numItems_ + 1,
    std::vector<int>(maxWeight_ + 1, 0));
65:     for (int i = 1; i <= numItems_; ++i)
66:     {
67:       for (int w = 1; w <= maxWeight_; ++w)
68:       {
69:         if (weights_[i - 1] <= w)
70:           dp[i][w] = std::max(dp[i - 1][w], dp[i - 1][w - weights_[i - 1]] +
    values_[i - 1]);
71:         else
72:           dp[i][w] = dp[i - 1][w];
73:       }
74:     }
75:     return dp[numItems_][maxWeight_];
76:   }
77:
78:   int Knapsack::knapsackBacktracking() const
79:   {
```

```
80:      int maxValue = 0;
81:      std::function<void(int, int, int)> backtrack = [&](int i, int
     currentWeight, int currentValue)
82:      {
83:        if (i == numItems_)
84:        {
85:          if (currentWeight <= maxWeight_)
86:            maxValue = std::max(maxValue, currentValue);
87:          return;
88:        }
89:        backtrack(i + 1, currentWeight, currentValue);
90:        backtrack(i + 1, currentWeight + weights_[i], currentValue +
     values_[i]);
91:      };
92:      backtrack(0, 0, 0);
93:      return maxValue;
94:    }
95:
96:    int Knapsack::knapsackBranchAndBound() const
97:    {
98:      struct Node
99:      {
100:       int level,
101:       profit,
102:       bound,
103:       weight;
104:     };
105:
106:     auto bound = [&](const Node& u)
107:     {
108:       if (u.weight >= maxWeight_)
109:         return 0;
110:       int profitBound = u.profit;
111:       int j = u.level + 1;
112:       int totWeight = u.weight;
113:       while ((j < numItems_) && (totWeight + weights_[j] <= maxWeight_))
114:       {
115:         totWeight += weights_[j];
116:         profitBound += values_[j];
117:         j++;
118:       }
119:       if (j < numItems_) profitBound += (maxWeight_ - totWeight) * values_[j]
     / weights_[j];
120:       return profitBound;
121:     };
122:
123:     std::vector<Node> Q;
124:     Node u, v;
125:     u.level = -1;
126:     u.profit = u.weight = 0;
127:     Q.push_back(u);
128:     int maxProfit = 0;
129:
```

```
130:       while (!Q.empty())
131:       {
132:         u = Q.back();
133:         Q.pop_back();
134:
135:         if (u.level == -1)
136:           v.level = 0;
137:         if (u.level == numItems_ - 1)
138:           continue;
139:
140:         v.level = u.level + 1;
141:         v.weight = u.weight + weights_[v.level];
142:         v.profit = u.profit + values_[v.level];
143:
144:         if (v.weight <= maxWeight_ && v.profit > maxProfit)
145:           maxProfit = v.profit;
146:
147:         v.bound = bound(v);
148:         if (v.bound > maxProfit)
149:           Q.push_back(v);
150:
151:         v.weight = u.weight;
152:         v.profit = u.profit;
153:         v.bound = bound(v);
154:         if (v.bound > maxProfit)
155:           Q.push_back(v);
156:       }
157:       return maxProfit;
158:     }
159:
160:     int Knapsack::knapsackBruteForce() const
161:     {
162:       int maxValue = 0;
163:       int numSubsets = 1 << numItems_;
164:       for (int subset = 0; subset < numSubsets; ++subset)
165:       {
166:         int totalWeight = 0, totalValue = 0;
167:         for (int i = 0; i < numItems_; ++i)
168:         {
169:           if (subset & (1 << i))
170:           {
171:             totalWeight += weights_[i];
172:             totalValue += values_[i];
173:           }
174:         }
175:         if (totalWeight <= maxWeight_)
176:         {
177:           maxValue = std::max(maxValue, totalValue);
178:         }
179:       }
180:       return maxValue;
181:     }
```

command.hpp

```
182:    #ifndef COMMAND_HPP
183:    #define COMMAND_HPP
184:
185:    #include <vector> // vector
186:    #include <string> // string
187:
188:    namespace command
189:    {
190:    void help();
191:    void create(int capacity, int numItems, const std::string& filename);
192:    void create(const std::string& filename);
193:    void show(const std::vector<std::string>& tokens);
194:    void solveDP(const std::string& filename);
195:    void solveBT(const std::string& filename);
196:    void solveBB(const std::string& filename);
197:    void solveBF(const std::string& filename);
198:    }
199:
200:    #endif
```

command.cpp

```
#include <iostream>
#include <fstream>  // ifstream, ofstream
#include <vector>  // vector
#include <random>  // rand
#include <memory>  // unique_ptr
#include <sstream>  // istringstream

#include "command.hpp"
#include "function.hpp"

using namespace kladkovoj;

std::unique_ptr<Knapsack> knapsack_;

void command::create(int capacity, int numItems, const std::string&
filename)
{
  std::ofstream file(filename);
  if (!file)
  {
    throw std::runtime_error("Could not create file.");
  }

  file << capacity << ' ' << numItems << '\n';
  for (int i = 0; i < numItems; ++i)
  {
    int weight = rand() % 100 + 1;
    int value = rand() % 100 + 1;
    file << weight << ' ' << value << '\n';
  }
}

void command::create(const std::string& filename)
{
  std::ofstream file(filename);
  if (!file)
  {
    throw std::runtime_error("Could not create file.");
  }

  int capacity, numItems;
  std::cout << "Enter knapsack capacity and number of items: ";
  std::cin >> capacity >> numItems;

  if (std::cin.fail())
  {
    throw std::runtime_error("Invalid input.");
  }

  file << capacity << ' ' << numItems << '\n';
  for (int i = 0; i < numItems; ++i)
```

```
251:       {
252:         int weight = rand() % 100 + 1;
253:         int value = rand() % 100 + 1;
254:         file << weight << ' ' << value << '\n';
255:       }
256:     }
257:
258:     void command::show(const std::vector<std::string>& tokens)
259:     {
260:       if (tokens.size() != 2)
261:       {
262:         throw std::invalid_argument("SHOW requires a filename.");
263:       }
264:       std::ifstream file(tokens[1]);
265:       if (!file)
266:       {
267:         throw std::runtime_error("Could not open file.");
268:       }
269:
270:       std::string firstLine;
271:       if (std::getline(file, firstLine))
272:       {
273:         std::istringstream iss(firstLine);
274:         std::size_t capacity, numItems;
275:         if (iss >> capacity >> numItems)
276:         {
277:           std::cout << "capacity: " << capacity << " number of items: " <<
      numItems << '\n';
278:         }
279:         else
280:         {
281:           throw std::runtime_error("Invalid format.");
282:         }
283:       }
284:
285:       std::size_t num = 1;
286:       std::string line;
287:       while (std::getline(file, line))
288:       {
289:         std::cout << num << ": " << line << '\n';
290:         num++;
291:       }
292:     }
293:
294:     void command::solveDP(const std::string& filename)
295:     {
296:       std::ifstream file(filename);
297:       if (!file)
298:       {
299:         throw std::runtime_error("Could not open file.");
300:       }
301:
302:       int capacity, numItems;
```

```
303:       file >> capacity >> numItems;
304:
305:       Knapsack knapsack(numItems, capacity);
306:       for (int i = 0; i < numItems; ++i)
307:       {
308:         int weight, value;
309:         file >> weight >> value;
310:         knapsack.addItem(weight, value);
311:       }
312:
313:       int result = knapsack.knapsackDP();
314:       std::cout << "Maximum value (DP): " << result << '\n';
315:     }
316:
317:     void command::solveBT(const std::string& filename)
318:     {
319:       std::ifstream file(filename);
320:       if (!file)
321:       {
322:         throw std::runtime_error("Could not open file.");
323:       }
324:
325:       int capacity, numItems;
326:       file >> capacity >> numItems;
327:
328:       Knapsack knapsack(numItems, capacity);
329:       for (int i = 0; i < numItems; ++i)
330:       {
331:         int weight, value;
332:         file >> weight >> value;
333:         knapsack.addItem(weight, value);
334:       }
335:
336:       int result = knapsack.knapsackBacktracking();
337:       std::cout << "Maximum value (BT): " << result << '\n';
338:     }
339:
340:     void command::solveBB(const std::string& filename)
341:     {
342:       std::ifstream file(filename);
343:       if (!file)
344:       {
345:         throw std::runtime_error("Could not open file.");
346:       }
347:
348:       int capacity, numItems;
349:       file >> capacity >> numItems;
350:
351:       Knapsack knapsack(numItems, capacity);
352:       for (int i = 0; i < numItems; ++i)
353:       {
354:         int weight, value;
355:         file >> weight >> value;
```

```
356:         knapsack.addItem(weight, value);
357:       }
358:
359:       int result = knapsack.knapsackBranchAndBound();
360:       std::cout << "Maximum value (BB): " << result << '\n';
361:     }
362:
363:     void command::solveBF(const std::string& filename)
364:     {
365:       std::ifstream file(filename);
366:       if (!file)
367:       {
368:         throw std::runtime_error("Could not open file.");
369:       }
370:
371:       int capacity, numItems;
372:       file >> capacity >> numItems;
373:
374:       Knapsack knapsack(numItems, capacity);
375:       for (int i = 0; i < numItems; ++i)
376:       {
377:         int weight, value;
378:         file >> weight >> value;
379:         knapsack.addItem(weight, value);
380:       }
381:
382:       int result = knapsack.knapsackBruteForce();
383:       std::cout << "Maximum value (BF): " << result << '\n';
384:     }
385:
386:
387:     void command::help()
388:     {
389:       std::cout << "Available commands:\n";
390:       std::cout << "HELP - Show this help message\n";
391:       std::cout << "CREATE <capacity> <numItems> <filename> - Create a knapsack
      with random items and save to file\n";
392:       std::cout << "CREATE <filename> - create a backpack from a file\n";
393:       std::cout << "SHOW <filename> - Show contents of the file\n";
394:       std::cout << "SOLVE_DP <filename> - Solve knapsack problem using dynamic
      programming\n";
395:       std::cout << "SOLVE_BT <filename> - Solve knapsack problem using
      backtracking\n";
396:       std::cout << "SOLVE_BB <filename> - Solve knapsack problem using branch
      and bound\n";
397:       std::cout << "SOLVE_BF <filename> - Solve knapsack problem using brute
      force\n";
398:     }
```

main.pp

```
399:    #include <iostream>
400:    #include <iterator>
401:    #include <limits>
402:    #include <map>
403:    #include <functional>
404:    #include <sstream>
405:
406:    #include "command.hpp"
407:    #include "function.hpp"
408:
409:    using namespace kladkovoj;
410:
411:    int main()
412:    {
413:      std::map<std::string, std::function<void(const
      std::vector<std::string>&)>> commandMap =
414:      {
415:        {"HELP", [](const std::vector<std::string>&)
416:        {
417:          command::help();
418:        }
419:        },
420:        {"CREATE", [](const std::vector<std::string>& tokens)
421:        {
422:          if(tokens.size() == 2)
423:          {
424:            command::create(tokens[1]);
425:          }
426:          else if(tokens.size() == 4)
427:          {
428:            int capacity = std::stoi(tokens[1]);
429:            int numItems = std::stoi(tokens[2]);
430:            command::create(capacity, numItems, tokens[3]);
431:          }
432:          else
433:          {
434:            std::cerr << "Invalid usage. Type HELP for a list of commands.\n";
435:          }
436:        }
437:        },
438:        {"SHOW", [](const std::vector<std::string>& tokens)
439:        {
440:          command::show(tokens);
441:        }
442:        },
443:        {"SOLVE_DP", [](const std::vector<std::string>& tokens)
444:        {
445:          if(tokens.size() == 2)
446:          {
447:            command::solveDP(tokens[1]);
448:          }
```

```
449:           else
450:           {
451:             throw std::invalid_argument("Requires a filename.");
452:           }
453:         }
454:         },
455:         {"SOLVE_BT", [](const std::vector<std::string>& tokens)
456:         {
457:           if(tokens.size() == 2)
458:           {
459:             command::solveBT(tokens[1]);
460:           }
461:           else
462:           {
463:             throw std::invalid_argument("Requires a filename.");
464:           }
465:         }
466:         },
467:         {"SOLVE_BB", [](const std::vector<std::string>& tokens)
468:         {
469:           if(tokens.size() == 2)
470:           {
471:             command::solveBB(tokens[1]);
472:           }
473:           else
474:           {
475:             throw std::invalid_argument("Requires a filename.");
476:           }
477:         }
478:         },
479:         {"SOLVE_BF", [](const std::vector<std::string>& tokens)
480:         {
481:           if(tokens.size() == 2)
482:           {
483:             command::solveBF(tokens[1]);
484:           }
485:           else
486:           {
487:             throw std::invalid_argument("Requires a filename.");
488:           }
489:         }
490:         }
491:       };
492:
493:       std::string line;
494:       while(std::getline(std::cin, line))
495:       {
496:         std::istringstream iss(line);
497:
498:         std::vector<std::string>
       tokens{std::istream_iterator<std::string>{iss},
       std::istream_iterator<std::string>{}};
499:         if(tokens.empty())
```

```
500:         {
501:           continue;
502:         }
503:
504:         try
505:         {
506:           auto it = commandMap.find(tokens[0]);
507:           if(it != commandMap.end())
508:           {
509:             it->second(tokens);
510:           }
511:           else if(!tokens[0].empty())
512:           {
513:             throw std::invalid_argument("<INVALID COMMAND>");
514:           }
515:         }
516:         catch(const std::exception& e)
517:         {
518:           std::cerr << e.what() << '\n';
519:         }
520:
521:         if (std::cin.fail())
522:         {
523:           std::cin.clear();
524:           std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
525:         }
526:       }
527:     }
```

Заключение

Программа позволяет эффективно решать задачу о рюкзаке с использованием различных методов. Реализация включает создание и чтение файлов с данными о рюкзаке и предметах, а также несколько алгоритмов для нахождения оптимального решения. Программа успешно прошла тестирование и показала корректные результаты для всех предоставленных тестовых случаев.