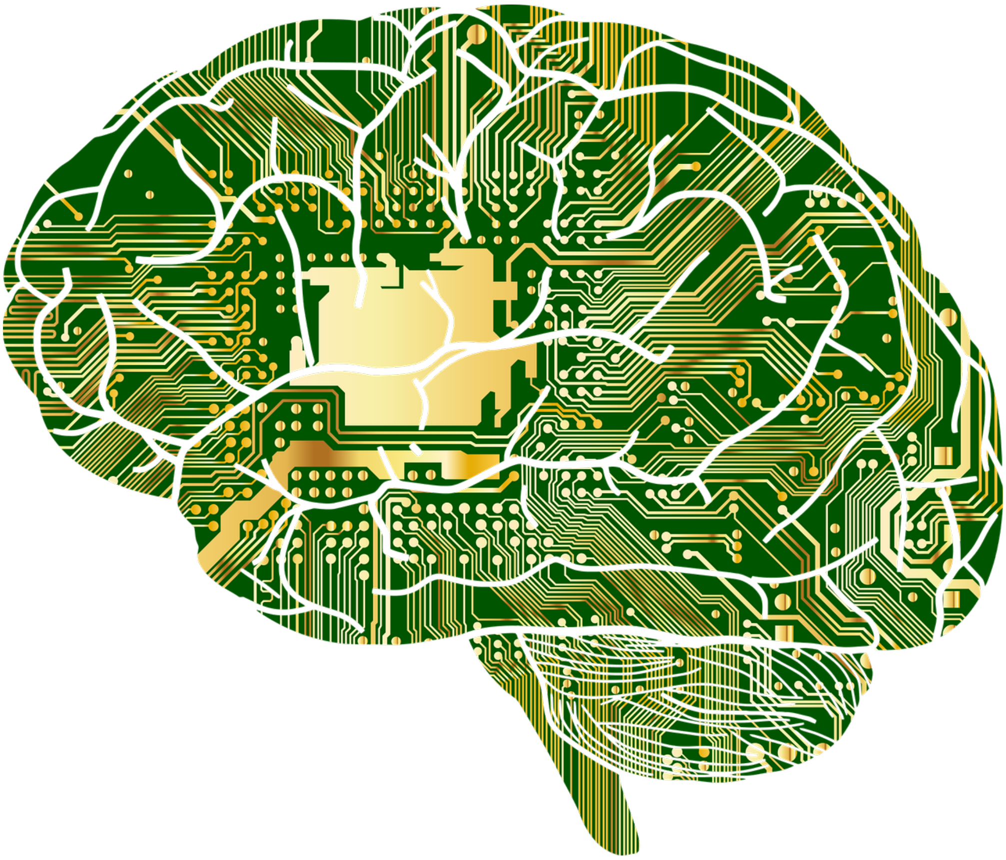
Laboratorium Sztucznej Inteligencji

Ćwiczenie 1. Prawdopodobieństwo i reguła Bayesa



Politechnika Poznańska

Instytut Robotyki i Inteligencji Maszynowej

Jan Wietrzykowski

Wstęp

Teoria prawdopodobieństwa jest podstawą działania wielu systemów sztucznej inteligencji, ponieważ umożliwia uwzględnienie niepewności. Pozwala ona na zdefiniowanie stopnia, w jakim traktujemy daną informację za prawdziwą, dysponując pewnymi obserwacjami. W tym ujęciu prawdopodobieństwo nie jest jedynie miarą, jak często pewne zjawisko występuje, ale jak bardzo ufamy, że dane wydarzenie zaszło lub zajdzie. Przykładem może być informacja o położeniu robota opisana za pomocą rozkładu prawdopodobieństwa, po uwzględnieniu odczytów ze skanera laserowego.

Stan świata w tym podejściu jest reprezentowany za pomocą par zmienna-wartość, a zmienne są nazwane zmiennymi losowymi. Każda zmienna losowa posiada swoją domenę, czyli zbiór wartości, które może przyjąć, np. zmienna losowa *Weather* może przyjmować

Podczas zajęć korzystać będziemy z następującej notacji:

- Prawdopodobieństwo, że zmienna losowa *Weather* przyjęła wartość *cloudy*. Jest to informacja o tym, jak często dane zjawisko występuje i reprezentuje naszą wiedzę wstępną o świecie (unconditional or prior probability):
$$P(Weather = cloudy) = P(cloudy) = 0.2$$
- Prawdopodobieństwo warunkowe, że zmienna losowa *Weather* przyjęła wartość *cloudy*, pod warunkiem, że zmienna losowa *Umbrella* przyjęła wartość *yes*. Reprezentuje nasze przekonanie, że świat znajduje się w określonym stanie, biorąc pod uwagę dokonane obserwacje (conditional or posterior probability):
$$P(Weather = cloudy | Umbrella = yes)$$
- Prawdopodobieństwo łączne, że zmienna losowa *Weather* przyjęła wartość *cloudy* i zmienna losowa *Umbrella* przyjęła wartość *yes*. Jest to informacja o tym, jak często dwa zdarzenia występują razem:
$$P(Weather = cloudy, Umbrella = yes)$$
- Rozkład prawdopodobieństwa zmiennej *Weather*. Zawiera informacje o prawdopodobieństwach dla wszystkich wartości danej zmiennej:
$$\mathbf{P}(Weather) = \langle 0.1, 0.1, 0.3, 0.5 \rangle$$
- Rozkład prawdopodobieństwa warunkowego zmiennej *Weather*, mając daną zmienną *Umbrella*. Ma on postać tabeli o wymiarach zdefiniowanych przez rozmiary domen zmiennych (w tym wypadku 3 x 2):
$$\mathbf{P}(Weather | Umbrella)$$

Użyteczne reguły:

- Reguła iloczynu:
$$P(a, b) = P(a|b)P(b)$$
- Obliczanie prawdopodobieństwa brzegowego:
$$\mathbf{P}(\mathbf{X}) = \sum_{\mathbf{y} \in \mathbf{Y}} \mathbf{P}(\mathbf{X}, \mathbf{y})$$
- Reguła Beyesa:
$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \alpha(a)P(a|b)P(b)$$
- Jeśli dwie zmienne są niezależne to:
$$P(a, b) = P(a|b)P(b) = P(a)P(b)$$
- Jeśli dwie zmienne są niezależne, mając daną trzecią zmienną to (nie jest to jednoznaczne z ogólną niezależnością zmiennych a i b):
$$P(a, b|c) = P(a|c)P(b|c)$$

Przebieg zajęć

Poniżej znajdują się zadania do rozwiązania podczas zajęć. Podczas ich rozwiązywania skorzystaj z PyCharma i szablonu kodu dostępnego poniżej.

```
#!/usr/bin/env python

"""code template"""

import numpy as np


def main():
    P = np.array([])
    print(P)

if __name__ == '__main__':
    main()
```

Zadanie 1

Naszym celem jest zdiagnozowanie pacjenta u dentysty. Chcemy się dowiedzieć czy ma on próchnicę (zmienna losowa *Cavity*), mając informację o tym czy boli go ząb (*Toothache*) i o tym czy podczas wiercenia zakleszczyło się wiertło dentystyczne (*Catch*). Łączny rozkład prawdopodobieństwa jest dany poniższą tabelą:

	$toothache$		$\sim toothache$	
	$catch$	$\sim catch$	$catch$	$\sim catch$
$cavity$	0.108	0.012	0.072	0.008
$\sim cavity$	0.016	0.064	0.144	0.576

1. Zapisz łączne prawdopodobieństwo jako tablicę *numpy* o odpowiednich wymiarach.
2. Oblicz $\mathbf{P}(Toothache)$ (przydatne funkcje: *np.sum*).

```
P_too =
[0.2 0.8]
```

3. Oblicz $\mathbf{P}(Cavity)$.

```
P_cav =
[0.2 0.8]
```

4. Oblicz $\mathbf{P}(Toothache|Cavity)$. Wynik zapisz tak, aby indeks zmiennej *Toothache* był pierwszym wymiarem (przydatne funkcje: *np.transpose*).

```
P_too_giv_cav =
[[0.6 0.1]
[0.4 0.9]]
```

5. [Oblicz $\mathbf{P}(Cavity|toothache \vee catch)$, gdzie $toothache \vee catch$ oznacza występowanie bólu zęba lub zakleszczenia.]
6. Jak zależy wielkość tablicy z pełnym rozkładem prawdopodobieństwa od liczby zmiennych, zakładając, że zmienne te są binarne (każda może przyjąć 2 wartości)?
7. Ile pamięci operacyjnej byłoby potrzebne do przechowania takiej tablicy dla 32 zmiennych, zapisując liczby jako 32 bitowy float?
8. Jak obliczyć $\mathbf{P}(Cavity|Toothache, Catch)$ nie znając pełnego rozkładu, a dysponując jedynie $\mathbf{P}(Toothache, Catch|Cavity)$ oraz $\mathbf{P}(Cavity)$? Zaimplementuj i przetestuj rozwiązanie (zasymuluj dostępność $\mathbf{P}(Toothache, Catch|Cavity)$ oraz $\mathbf{P}(Cavity)$ obliczając te rozkłady z rozkładu łącznego). Jakie jest prawdopodobieństwo, że pacjent ma próchnicę, jeśli boli go ząb i wiertło nie zakleszczyło się w zębie? A jakie jeśli boli go ząb i wiertło zakleszczyło się w zębie? Rozkład powinien wyglądać następująco:

```
P_cav_giv_too_cat =
[[[0.87096774 0.15789474]
  [0.33333333 0.01369863]]

 [[0.12903226 0.84210526]
  [0.66666667 0.98630137]]]
```

9. Czy zmienne *Toothache* i *Catch* są od siebie niezależne? Co z niezależnością warunkową, mając dane *Cavity*?
10. Wykorzystaj te zależności, aby obliczyć $\mathbf{P}(Cavity|Toothache, Catch)$ mając dane $\mathbf{P}(Toothache|Cavity), \mathbf{P}(Catch|Cavity)$ oraz $\mathbf{P}(Cavity)$.
11. Jak rozłożyć pełen rozkład prawdopodobieństwa za pomocą danych z poprzedniego podpunktu?
12. Ile pamięci potrzeba do przechowywania pełnego rozkładu, rozłożonego na czynniki, jeśli mamy 31 niezależnych warunkowo zmiennych i jedną zmienną separującą te zmienne?

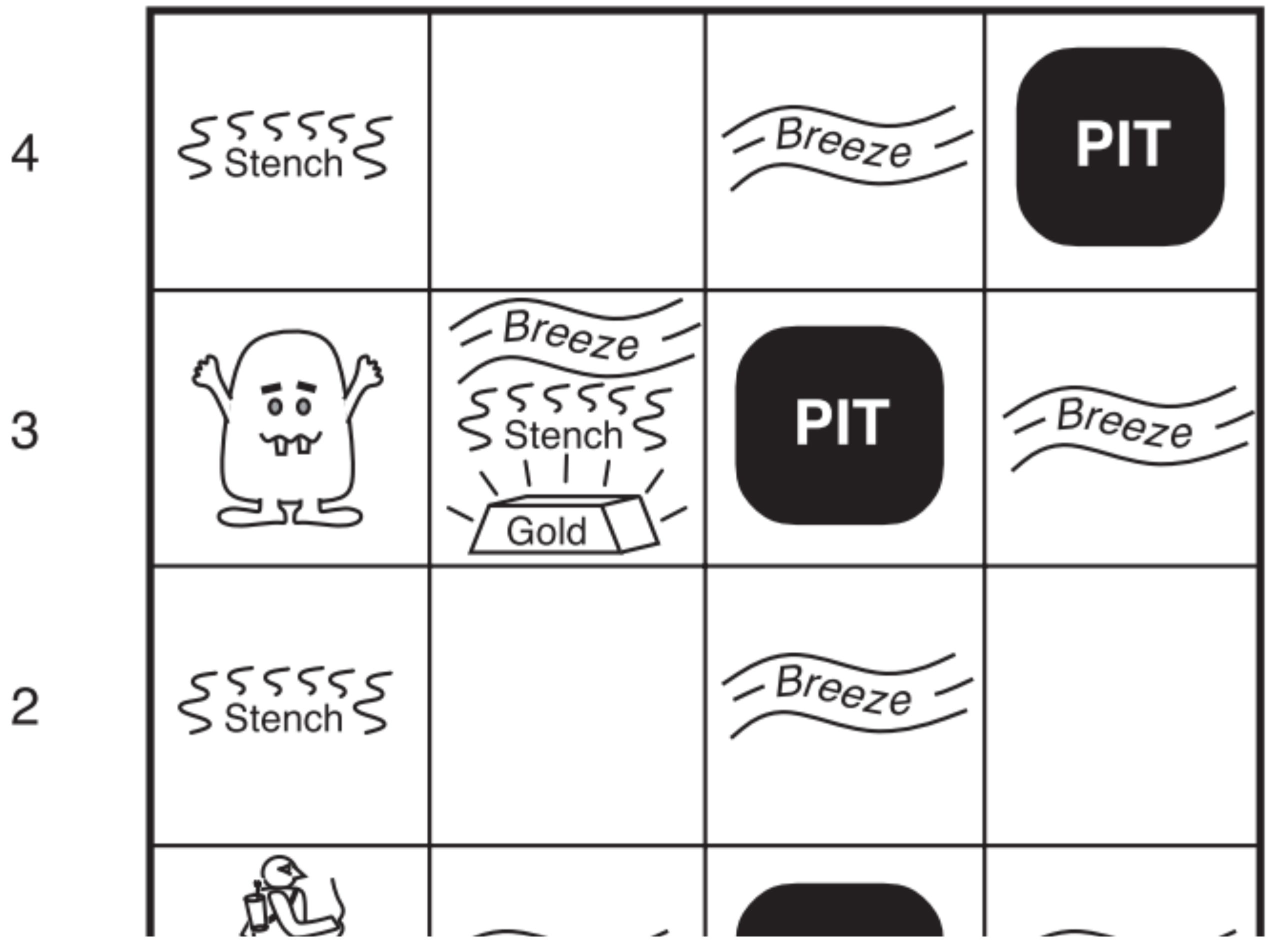
Zadanie 1 - odpowiedzi

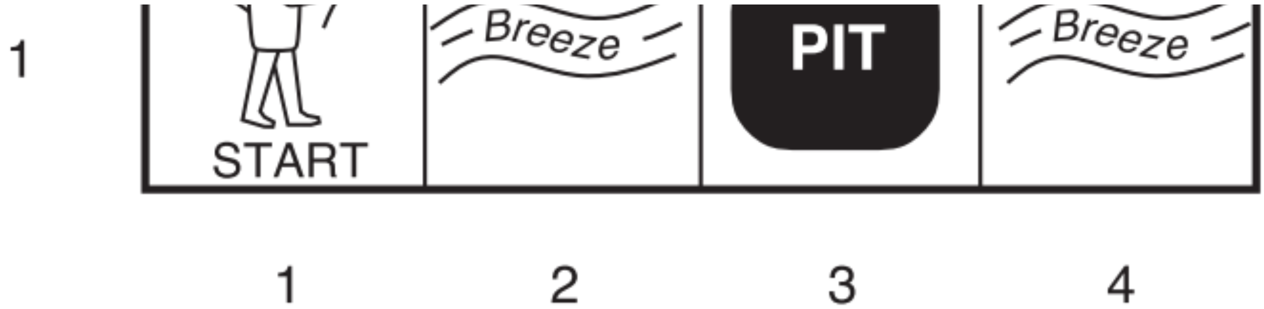
Zadania dodatkowe

- Russell, Norvig, ex. 13.13
- Russell, Norvig, ex. 13.15

Zadanie 2

Masz za zadanie napisać algorytm dla eksploratora jaskiń w Świecie Wumpusa. Celem eksploratora jest odwiedzenie jak największej liczby lokacji przed wpadnięciem do dołu. Świat ten ma postać regularnej siatki o wymiarach 4 x 4. Każda lokacja może zawierać dół z prawdopodobieństwem 0.2. Jeśli eksplorator wejdzie do takiej lokacji, od razu ginie. W aktualnie rozpatrywanym świecie ignorujemy samego Wumpusa oraz złoto. Eksplorator może poruszać się do przodu w aktualnym kierunku, obrócić się w lewo i obrócić się w prawo (są 4 możliwe kierunki eksloratora). Jeśli eksplorator znajdzie się w lokacji sąsiadujacej z dołem (lokacje muszą sąsiadować całą krawędzią, więc każda lokacja ma co najwyżej 4 sąsiadów), poczuje on podmuch. Poniżej pokazano przykładowy świat.

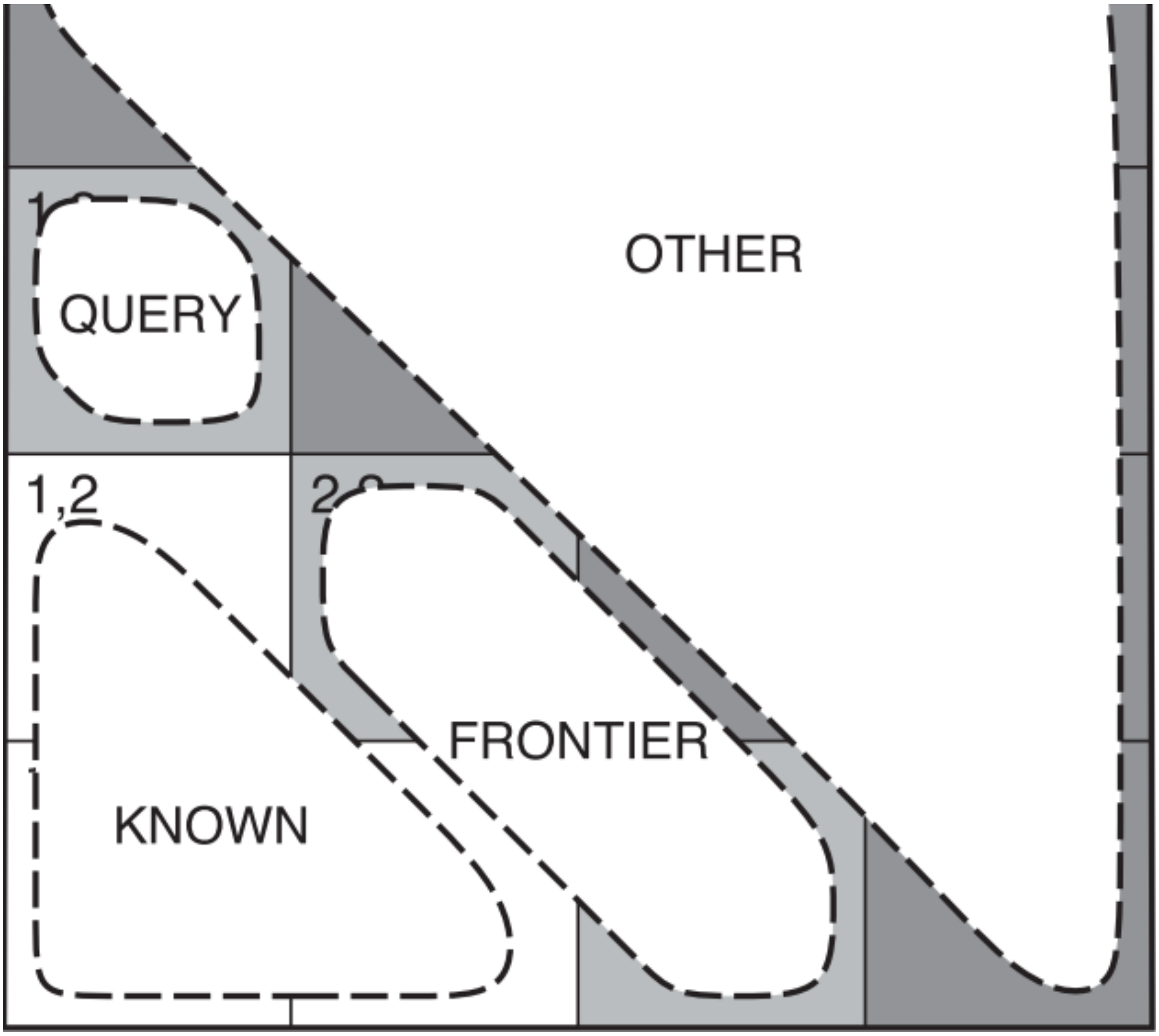




Na początku gry eksplorator znajduje się w lokacji (0, 0) i nie wie, co znajduje się w pozostałych lokacjach. Za każdym razem, kiedy trafi do nowej lokacji, jest w stanie wyczuć obecność podmuchu lub jego brak. Chcielibyśmy obliczać jakie jest prawdopodobieństwo, że w lokacjach sąsiadujących z dotychczas odwiedzonymi znajdują się doły i wybrać tę, dla której prawdopodobieństwo to jest najmniejsze. W tym celu wprowadzimy zmienne od $Pit_{0,0}$ do $Pit_{3,3}$ przyjmujące wartości prawda/fałsz i oznaczające obecność dołu lub jego brak oraz zmienne $Bre_{0,0}$ do $Bre_{3,3}$ oznaczające obecność powiewu lub jego brak.

1. Łączny rozkład prawdopodobieństwa $\mathbf{P}(Pit_{0,0}, \ldots, Pit_{3,3}, Bre_0, \ldots, Bre_{n-1})$, gdzie Bre_i oznaczją obecność podmuchu w już odwiedzonych lokacjach, możemy rozłożyć za pomocą reguły iloczynu na rozkład warunkowy dla podmuchu oraz *prior* dla dołów:
   $\mathbf{P}(Pit_{0,0}, \ldots, Pit_{3,3}, Bre_0, \ldots, Bre_{n-1}) = \mathbf{P}(Bre_0, \ldots, Bre_{n-1} | Pit_{0,0}, \ldots, Pi$
2. Ile wynosi wartość $\mathbf{P}(Bre_0, \ldots, Bre_{n-1} | Pit_{0,0}, \ldots, Pit_{3,3})$ jeśli podmuch jest tylko w lokacjach sąsiadujących z dołami? A ile jeśli rozkład podmuchów i dołów nie zgadza się z regułami gry?
3. Jak można dalej rozłożyć $\mathbf{P}(Pit_{0,0}, \ldots, Pit_{3,3})$ korzystając z niezależności?
4. Podzielmy zmienne na kilka zbiorów, które pozwolą nam na łatwiejszą manipulację wzorami:
   - *Known* (*Kno*) - zbiór zmiennych *Pit* zawierający już odwiedzone lokacje.
   - *Query* (*Q*) - zmienna *Pit* związana z lokacją, dla której chcemy obliczyć prawdopodobieństwo zawierania dołu.
   - *Frontier* (*Fro*) - zbiór zmiennych *Pit* dla wszystkich lokacji sąsiadujących z już odwiedzonymi.
   - *Other* (*Oth*) - zbiór zmiennych *Pit* dla wszystkich lokacji jeszcze nie odwiedzony i nie sąsiadujących z już odwiedzonym (pola poza naszym zasięgiem eksploracji).
   - *Unknown* (*Unk*) - suma zbiorów *Fro* i *Oth*.
   - *Breeze* (*Bre*) - zbiór zmiennych *Bre* dla już odwiedzonych lokacji.

   Poniżej przykład dla odwiedzonych pól (0, 0), (1, 0) i (0, 1) i zapytania o pole (0, 2).



Jak obliczyć $\mathbf{P}(Q|kno, bre)$? Wykorzystaj regułę iloczynu i wzór na obliczanie prawdopodobieństwa brzegowego z rozkładu łącznego (zmarginalizuj wszystkie zmienne *Unk*).

5. Jaka jest złożoność obliczeniowa otrzymanego wzoru względem liczby zmiennych *Unk*?

6. Wykorzystaj [szablon kodu]() i napisz algorytm, który wykorzysta ten wzór do sterowania agentem. Kod do uzupełnienia znajduje się w pliku *agents/prob.py*. Funkcja obliczająca $\mathbf{P}(Q|kno, bre)$ została już zaimplementowana jako metoda *prob_bf*. Znajdź następną lokację do odwiedzenia wśród lokacji sąsiadujących z już odwiedzonymi (zmienna *self.front*), dla której prawdopodobieństwo wpadnięcia do dołu jest najmniejsze. Wykorzystaj metodę *path_to_loc* do zaplanowania ciągu komend, które zaprowadzą eksploratora do wybranej lokacji. Jeśli więcej niż jedna lokacja ma najniższe prawdopodobieństwo, wybierz tę, do której można się najszybciej dostać (potrzeba najmniej komend).

   Wskazówki:

   - Otwórz projekt w PyCharmie i skonfiguruj opcje uruchamiania, aby uruchamiać odpowiedni skrypt (*main.py*).

7. Czy można w bardziej efektywny sposób obliczyć $\mathbf{P}(Q|kno, bre)$? Intuicyjnie, czy zmienne *Oth* wpływają na prawdopodobieństwo wystąpienia dołu w lokacjach sąsiadujących z już odwiedzonymi? Rozłóż wzór z punktu 4. na dwie sumy - dla zmiennych *Fro* i *Oth*. Czy część warunkowa zależy od *Oth*? Czy można umieścić teraz sumę po *Oth* w innym miejscu?
8. Pozbądź się stałych i powyłączaj przed sumy jak najwięcej czynników. Wykorzystaj właściwość, że sumowanie prawdopodobieństwa po wszystkich możliwych kombinacjach zmiennych daje w wyniku 1.
9. Zaimplementuj rozwiązanie i sprawdź szybkość jego działania w porównaniu do poprzedniego rozwiązania. Sprawdź czy otrzymujesz takie same rezultaty w obu podejściach.
10. Czy stała α' będzie dla każdej lokacji taka sama? Czy to zmienia coś w kwestii ilości potrzebnych obliczeń, jeśli naszym celem jest jedynie wybór lokacji z najmniejszym prawdopodobieństwem?

Zadanie 2 - odpowiedzi

1.
2. Jeśli rozkład podmuchów i dołów się zgadza to prawdopodobieństwo wynosi 1, natomiast w przeciwnym wypadku 0. Wynika to z zasad gry, ponieważ zawsze przy dole będzie odczuwalny podmuch.
3. Zmienne *Pit* są niezależne, więc:

$$\mathbf{P}(Pit_{0,0}, \ldots, Pit_{3,3}) = \prod_{i,j=0,0}^{3,3} \mathbf{P}(Pit_{i,j})$$

4. Korzystając najpierw z reguły iloczynu, a później z marginalizacji:

$$\mathbf{P}(Q|kno, bre) = \frac{\mathbf{P}(Q, kno, bre)}{P(kno, bre)} = \alpha\mathbf{P}(Q, kno, bre) = \alpha \sum_{unk} \mathbf{P}(Q, kno, unk,$$

   Następnie wykorzystując wzór z punktu 1. i 3.:

$$\alpha \sum_{unk} \mathbf{P}(Q, kno, unk, bre) = \alpha \sum_{unk} P(bre|kno, unk, Q)\mathbf{P}(Q)P(kno)P(unk)$$

5. Złożoność to $2^{|Unk|}$.
6. Aby znaleźć lokacje z najmniejszym prawdopodobieństwem wpadnięcia do dołu trzeba przejrzeć wszystkie elementy w słowniku *front_to_prob*:

```
best_dests = [self.loc]
```

```
best_dests_prob = 2.0
for dest, prob in front_to_prob.items():
   # if new probability very close to the best one so far
   if abs(prob - best_dests_prob) < 1e-5:
       # add to list
       best_dests.append(dest)
   # if new probability is better
   elif prob < best_dests_prob:
       # empty the list and add new element
       best_dests = [dest]
       best_dests_prob = prob
```

Następnie wystarczy znaleźć tę lokację, do której ścieżka jest najkrótsza:

```
best_dest = self.loc
best_dest_cmds_len = 1e6
best_dest_cmds = ['forward']
for dest in best_dests:
   # find path to location
   cmds = self.path_to_loc(dest)
   # if path shorter than the shortest so far
   if len(cmds) < best_dest_cmds_len:
       best_dest = dest
       best_dest_cmds_len = len(cmds)
       best_dest_cmds = cmds
```

7. Rozkład na dwie sumy:

$$\alpha \sum_{unk} P(bre|kno, unk, Q)\mathbf{P}(Q)P(kno)P(unk) = \alpha \sum_{fro} \sum_{oth} P(bre|kno, fro, ot$$

Część warunkowa nie zależy od *oth*:

$$\alpha \sum_{fro} \sum_{oth} P(bre|kno, fro, Q)\mathbf{P}(Q)P(kno)P(fro)P(oth) = \alpha \sum_{fro} P(bre|kno,$$

8. Suma $\sum_{oth} P(oth) = 1$:

$$\alpha \mathbf{P}(Q)P(kno) \sum_{fro} P(bre|kno, fro, Q)P(fro)$$

Wartość $P(kno)$ jest stała, więc można ją włączyć do α:

$$\alpha' \mathbf{P}(Q) \sum_{fro} P(bre|kno, fro, Q)P(fro)$$

Wartości $\mathbf{P}(Q)$ i $P(fro)$ znamy, ponieważ wynikają one z *prior* dla dołów, natomiast $P(bre|kno, fro, Q)$ jest równe 0 lub 1 w zależności od tego czy rozkład dołów i podmuchów się zgadza.

9. Na podstawie kodu z metody *prob_bf*:

```
front_to_prob = {}
for cur_loc in self.front:
   # set of frontier locations minus current query location
   front_other = self.front.copy()
   front_other.remove(cur_loc)

   # unnormalized probabilities of query containing pit (q) given known
   P_q_giv_k_b = 0
   # unnormalized probabilities of query not containing pit (nq) given
   P_nq_giv_k_b = 0
   # all combinations of pits for all possible numbers of pits
   for num_pits in range(len(front_other) + 1):
       for cur_pits in combinations(front_other, num_pits):
           # if query contains pit then add it to set of pits
           pits_q = set(cur_pits).union({cur_loc})
           pits_nq = set(cur_pits)

           # probability of breeze givern known, query and frontier
           P_b_giv_k_q_f = self.check_breeze(pits_q, self.vis)
           P_b_giv_k_nq_f = self.check_breeze(pits_nq, self.vis)
           # multiply by prior probability of given pit configuration
           P_q_giv_k_b += P_b_giv_k_q_f * \
                           self.pit_prob ** len(pits_q) * \
                           (1 - self.pit_prob) ** (len(self.front) - le
           P_nq_giv_k_b += P_b_giv_k_nq_f * \
                           self.pit_prob ** len(pits_nq) * \
                           (1 - self.pit_prob) ** (len(self.front) - le

   print('sum = ', (P_q_giv_k_b + P_nq_giv_k_b))
   # normalize, so sum is equal to 1
   P_q_giv_k_b_norm = P_q_giv_k_b / (P_q_giv_k_b + P_nq_giv_k_b)
   front_to_prob[cur_loc] = P_q_giv_k_b_norm
```

10. Suma będzie dla każdej lokacji taka sama, ponieważ zależy tylko od *kno* i *bre*. Można więc pominać normalizację, ponieważ do wyboru lokacji potrzebujemy znaleźć wartość

minimalną, na co α' nie ma wpływu.